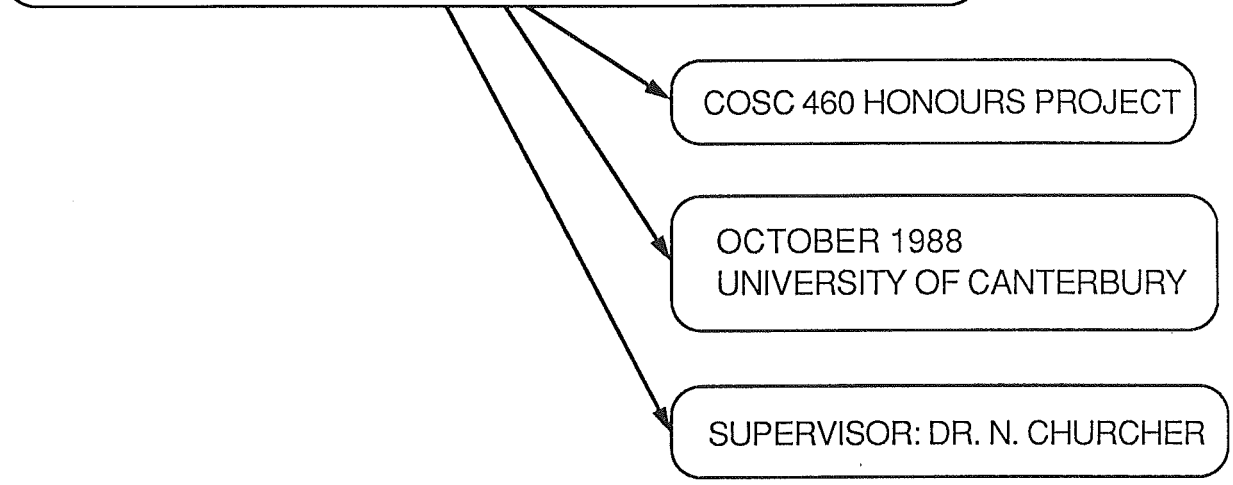
A SURVEY OF A DATABASE DESIGN TECHNIQUE

STEPHEN NEIL



FOREWORD

This project was an ambitious undertaking. There was a high probability of a lot of work been done, for little or no reward. This would appear to be a property of research projects that actually entail research in its pure form.

Fortunately, the results of the research that was performed in this project has proven to be rewarding, with numerous topics that are suitable to appear in a publication on the research subject. It is hoped that some recognition of the fact that the author has tried to enter into the spirit of a research project ,and actually tried to perform "research", will be given.

TABLE OF CONTENTS Page

PART	i	A General introduction to the Project.	1
	i.1	Collaboration with the CDB.	2
	i.2	Dependency List Synthesis.	2
	i.3	Background information on the topics to be covered.	3
	i.3.1	Introduction.	3
	i.3.2	History of Data Bases and Data management.	3
	i.3.3	Data modelling.	4
	i.3.3.1	Relational.	5
	i.3.3.2	File systems.	5
	i.4	Define Schema.	5
	i.5	Academic interest in design techniques.	7
	i.6	Purpose and aims of the Project.	7
PART	1	DLS UNDER THE ACAMEDIC MICROSCOPE.	8
	1.1	Introduction.	8
	1.1.1	Aims of this part of the project.	8
	1.1.2	Semantic content of the ER model.	8
Section	1.2	The Research results.	9
	1.2.1	Introductory section.	9
Section	1.2.2	Attributes.	9
	1.2.2.1	Introduction.	9
	1.2.2.2	Original E-R definition of an attribute.	10
	1.2.2.3	Proposal.	11
	1.2.2.4	Discussion.	11
	1.2.2.5	Future applications - Use of the above.	14
Section	1.2.3	Multivalued dependencies.	15
	1.2.3.1	Introduction.	15
	1.2.3.2	Definitions.	15
	1.2.3.3	Discussion.	16
	1.2.3.3.1	Example cases of MVDs.	16
	1.2.3.3.2	Example 1.	17

	1.2.3.3.3	Example 2.	18
	1.2.3.3.4	The differences between the approaches that use MVD definitions.	20
	1.2.3.4	MVDs in relations with only 2 attributes.	21
	1.2.3.5	Conclusions.	22
Section	1.2.4	Semantic Constructs.	22
	1.2.4.1	Introduction.	22
	1.2.4.2	Hierarchy definition.	23
	1.2.4.3	Discussion.	24
	1.2.4.4	Conclusions.	25
Section	1.2.5	Fan-Traps.	26
	1.2.5.1	Introduction.	26
	1.2.5.2	Ordinary Fan-traps.	26
	1.2.5.2.1	Description.	26
	1.2.5.2.2	Discussion.	27
	1.2.5.2.3	Recommendations for this case.	28
	1.2.5.3	Complex Fan-traps.	29
	1.2.5.4	Conclusions.	31
Section	1.2.6	Normal forms.	32
	1.2.6.1	Introduction.	32
	1.2.6.2	1NF.	32
	1.2.6.3	2NF.	33
	1.2.6.3	3NF.	34
	1.2.6.4	Discussion.	34
	1.2.6.4.2	Transitive Dependence.	35
	1.2.6.5	BCNF.	35
	1.2.6.5.1	Introduction.	35
	1.2.6.5.2	DLS performance w.r.t BCNF.	36
	1.2.6.5.3	Investigation into correcting problems.	38
	1.2.6.5.4	How to identify problem cases.	40

	1.2.6.6	4NF.	41
	1.2.6.6.1	Introduction.	41
	1.2.6.6.2	Discussion.	41
	1.2.6.6.3	4NF as it relates to fan-traps.	42
	1.2.7.6	5NF.	43
	1.2.7.6.1	Introduction.	43
	1.2.6.7.2	5NF as it relates to n-dependency and dependency information.	43
	1.2.6.8	Conclusions of Normal form investigation.	43
Section	1.2.7	Optionality, (or membership class).	44
	1.2.7.1	Introduction.	44
	1.2.7.2	Discussion.	44
	1.2.7.3	Conclusions.	45
Section	1.2.8	N-ary relationships.	45
	1.2.8.1	Introduction.	45
	1.2.8.2	Binary relationships.	46
	1.2.8.2.1	Introduction.	46
	1.2.8.2.2	DLS performance w.r.t binary.	46
	1.2.8.2.3	Problems illustrated.	48
	1.2.8.3	Ternary relationships	49
	1.2.8.3.1	Introduction.	49
	1.2.8.3.2	DLS performance evaluation w.r.t ternary.	49
	1.2.8.4	Identifying n-ary relationships.	51
	1.2.8.5	Discussion of problems.	52
	1.2.8.6	Proposals.	54
	1.2.8.7	Attributes of n-dependencies.	54
Section	1.3.0	Summary.	55
	1.3.1	Useful guidelines to the use of DLS.	55
	1.3.2	Academic discoveries made in DLS.	55

PART 2: A real life application. 56

2.1 Introduction. 56
2.1.1 The problem. 56
2.1.2 Schema conversion issues. 56

Section 2.2 The analysis. 58
2.2.1 The system to be analysed. 58
2.2.2 The analysis performed. 59
2.2.2.1 CDB's data processing methods. 59
2.2.2.2 Details of the analysis. 59
2.2.2.3 The Bottom-up analysis. 60
2.2.2.4 Top-down analysis: the modular structure. 61
2.2.3 The resulting dependency lists and diagrams. 61
2.2.4 Areas of interest in the analysis. 61
2.2.4.1 Attribute identification. 61
2.2.4.2 Correct dependency identification. 62
2.2.5 Results of the analysis. 62

Section 2.3 Results and conclusions. 63
2.3.1 CDB's benefits. 63
2.3.2 DLS's analysis benefits. 63

Part 3 Summary. 64

References. 65

Appendix A The General Ledger module. 69
Appendix B The Creditors Module. 75
Appendix C The Debtors module. 78
Appendix D The Stock module. 83

PART i A General introduction to the Project.

This project is primarily concerned with the area of Database schema design methods. This is an area of computer science that is the focus of much current research and development, both in the academic world and in the commercial marketplace. Often there is a significant difference between what the academic principles recommend and how a commercial application will proceed.

There are many papers and books that have been published on schema design methods and the larger topic of data modelling, (for example [Howe 1984]). These techniques have been useful in defining and refining the data modelling process. Unfortunately many of them have had little practical significance in "real world" applications.

"In short, data models offer abstractions of computer phenomena involving files and computer processing, rather than abstractions of real world phenomena. The consequences is that the distance between the application, as perceived by its human agents, and a data model, leading to a computer implementation, is too wide." [Furtado 1986]

Both formal and informal design methods exist and both types have problems: the formal techniques suffer from NP complete problems, and informal techniques often produce non-normalised databases. Some of the problems only become apparent until a "real" problem is attempted, especially the NP complete "tractability" problem.

Therefore a thorough investigation into the usefulness of a design method has been attempted. This investigation takes into account both: the formality of the technique and the product that it produces, and its usefulness when applied to a "real-life" application.

Such topics as the complexity of the modelling process, the ease of use, the "relational normalisation" of the finished result, and how well the method copes with semantic modelling problems in schema design will be dealt with.

The project is split into two main parts:

- the first is concerned with an academic investigation of the diagramming technique, and
- the second concerned with the performance of the technique when confronted with a real world problem.

The first part takes several issues that have been raised elsewhere, and applies them directly to the method. As a result several advancements to the method, and several useful guidelines to the use of the method are proposed. The issues that are dealt with in this part are as follows: the usefulness of defining attributes in this method (1.2.2), the DLS definition of MVDs (1.2.3), how DLS handles different semantic constructs (1.2.4), fan-traps (1.2.5), the level of normalisation that DLS produces (1.2.6), the omission of optional relationships in DLS, and n-ary relationships (1.2.8).

The second part is applying the design method on an application that it has never been tried on before. The application is "schema conversion" and "integration"; to the authors knowledge the method has only been used for design purposes.

The part is broken down into 3 sections: section 2.1 introduces the relevant issues, section 2.2 details the analysis, and section 2.3 presents the results.

i.1 Collaboration with the CDB.

This project involves collaboration with the EDP department of the Christchurch Drainage Board (CDB). They currently have an information management system which consists of a suite of Basic programs containing calls to the User 11 file system, (on a PDP-11/45 machine), for their day to day processing requirements. They recently acquired a MICROVAX 11, which will allow the transfer of much of their processing needs to this new machine. It is intended to use the relational database rdb system as a replacement for the existing User 11 system.

This conversion process provides a "real world" application. To design a suitable rdb database has proved to be an extensive job.

i.2 Dependency List Synthesis.

The schema design method that will be examined is the new method proposed by H. Smith in "Database Design: composing fully normalised tables from a rigorous dependency diagram" [Smith 1985].

This technique relies a lot less on mathematical formalisms than many previous techniques do, and is aimed at users who possibly do not know, or cannot cope with, many of the formal database theories. This technique will be referred to as **DLS**. As DLS is new not many documented applications of the technique are available in the literature. The only documented reference known to the author of DLS being used in an application is [Van Roessel 1987]. Though DLS was very useful in this instance, it did not push the technique to its limits, indeed no schema design method would be expected to have trouble in deriving a suitable representation for this problem.

I therefore consider DLS to be an unproven technique, with a lot of scope for examination.

i.3 Background information on the topics to be covered.

i.3.1 Introduction.

Before starting the investigation several terms and techniques must be introduced and defined.

i.3.2 History of Data Bases and Data management.

Since the late 1960's there has been a shift to Database solutions to file system problems. "A Database itself can be regarded as a kind of electronic filing cabinet" [Date 1986]. The function of a database is to maintain information and to make that information available on demand. The advantage of Databases, when used correctly over a more traditional paper-based method can be illustrated in these few practical examples, (this is by no means a comprehensive list):

- A computer will, (now-days), be much more compact than a large system of paper records.
- A database system will be able to effect changes in the database far more quickly than a paper-based system. In fact in on-line systems the database will be changed to reflect the new state of the real world almost instantaneously.
- Due to the above speed, the data in a database will represent the true state of the system that it is modelling much more regularly than its paper-based counterpart.
- Maintaining a paper based system can be a very laborious task. An electronic database can eliminate much of the tedium associated with maintaining an information system.
- A database has centralised control of the enterprise, whereas paper-based system tend to be compartmentalised and therefore information is widely dispersed. This centralised control offers many advantages: concurrency of application, reduction in redundancy, recovery of failure, to name but a few.

i.3.3 Data modelling.

A description of data that is independent of any Database Management System, (DBMS), software is referred to as a data model. The concept of a data model was first formulated by Codd, [Codd 1970], in his classic 1970 paper that is the original formulation of the relational model.
There are five main types of database models:

Inverted list,
Hierarchic,
Network,
Relational, and
Object Orientated Database Systems, (OODDS).

OODBS is a very new data model. The newest of the remaining four is the Relational model. The other three models were not developed on the basis of any predefined data abstracts, and any formalising of these models, (as has been done for the Network and Hierarchic models), were done after the fact. Also they are all at a lower level of abstraction than the relational model. This is illustrated in [Chen 1976] which is the original definition of the Entity-Relationship model, (this paper only deals with the relational and Network models, even at the time of publishing, 1976, the other 2 models were considered redundant and not worthwhile including in the analysis). Chen's paper provides four "levels of logical views" defined in a hierarchy:

- "level 1: Information concerning Entities and Relationships,
- level 2: Information structure,
- level 3: Access - path - independent data structure,
- level 4: Access - path - dependent data structure."

Chen also states that "the relational model is mainly concerned with levels 2 and 3" and that "the Network model is mainly concerned with level 4". Examining Dates description of the inverted list model, it would appear that it is primarily concerned with levels 3 and 4. Furthermore the hierarchic model would appear to mainly deal with level 4.

i.3.3.1 Relational.

"In late 1968 the principles of the relational model were laid down by one man, Dr E. F. Codd, a member of the IBM San Jose Research Laboratory. Codd, a mathematician, first realized that the discipline of mathematics could be used to inject some solid principles and rigor into database management that prior to that time was very deficient in these qualities." [Date 1986]

Though both hierarchical and network implementations existed prior to that time the hierarchic and network model were not formulated until later. By contrast, relational implementations did not appear until after the formulation of the data model. This provided a major advantage for the relational model - like being born with a silver spoon in your mouth.

Almost all the database systems developed over the past few years are Relational, and almost all current database research is also based on the relational model. The OODBS will be limited to future application sections, because it is so new that a standardised model has not yet been developed.

All developments that are proposed in this paper will be dealing with the relational model and can be assumed unless otherwise explicitly stated.

i.3.3.2 File systems.

Another method for storing and retrieving data are "File systems". These are not database system and follow none of the aforementioned database models for the problems solved by DBMS. Thus they are usually of little interest. The reason that they are mentioned here is that a particular example of a file system will be used in Part 2.

i.4 Define Schema.

Given a data model, (in this case the relational model), the next step in designing and implementing a DBMS is defining the Schema of the data. This is essentially the topic that the main thrust of this paper is aimed at.

The schema relates to the architecture of a database. The ANSI/SPARC has divided the architecture into three general levels [ANSI/SPARC 1978]:

- External, including a simplified model of the real world as seen by one or more application.
- Conceptual, including the limited model of the real world maintained for all applications of the enterprise.
- Internal, including a model of the data maintained for the representation of this limited model of the real world.

The external realm, ("realm" as used in [ANSI/SPARC 1978]), contains any number of external views of the database, each of which is a collection of objects representing the entities, properties, and relationships of interest to a specific application. Each external view of the database is associated with an external schema describing the objects in that external view of the database.

Similarly, for the conceptual realm there is a conceptual view of the database. This is a collection of objects representing the entities, properties, and relationships of interest to the entire enterprise. The descriptions of the objects is called the conceptual schema.

Finally, for the internal realm there is the internal view of the database. It is described by the internal schema of the database. This view deals with the internal representation of the data in the machine that the database will be implemented on.

It is apparent that "schema" generally means the definition of a view of the database. The term schema by itself will mean the "data definition" or "data description" of the enterprise. The external schema will not be dealt with very often, for one external schema will quite often be different from another as the views of the database differ. The task that will be undertaken will generally be to define the conceptual schema, and will be assumed unless otherwise stated.

It is helpful to view schema as properties of data that are true for all time. When dealing with a tuple in an application of the relational model it is useful to further consider the schema as being the entries in the column names and not the row names.

Database design can therefore be renamed schema design.

i.5 Academic interest in design techniques

The process of database schema design is a topic of current academic interest. It is an area that has real life applications, but often the downfalls of current academic techniques outweigh the benefits when applied to a common application.

There has been much effort devoted to the formalisation of the techniques used in designing and maintaining a DBMS. Unfortunately there seems to be a gap between what academic principles recommend - in the form of design methodologies - and what is actually used in the commercial environment, in the form of actual databases on real world machines. "Most methodologies were developed as parts of research projects with low emphasis on developing full-scale automated systems" [Batini etal. 1986].

Different design methods will produce different schemas, so how is a good design measured, and what are the benefits and consequences of a good or bad design?

i.6 Purpose and aims of the Project

A good technique will produce a schema that will be able to: minimise complexity, be extendible, allow schema integration, and allow evolution of the database.

The purpose of this project is to gauge some level of effectiveness of an existing design methodology. Furthermore, it is envisaged the identification of points where this technique has inadequacies will be possible. This will enable some actual experimental modification of the technique, with the aim of removing some of the problem areas.

Before continuing to Part 1 it is recommended that the reader is familiar with the terms derived in the accompanied Technical Document.

PART 1: DLS UNDER THE ACAMEDIC MICROSCOPE.

1.1 Introduction.

1.1.1 Aims of this part of the project.

As previously stated in section 3.4 of the associated Technical Report it is my conclusion that DLS parallels Kent's fact-based model. But DLS has omitted many issues that are catered for in the fact-based model. Also there are several outstanding issues that are raised when comparing DLS with other design techniques. These issues include:

- not following the standard definition of MVDs, and
- not providing specific ways of representing semantic constructs, and
- not mentioning BCNF in the "Diagramming conventions applied to normal-form guidelines" section in the original paper, and
- not mentioning, and providing for, n-ary relationships.

DLS is presented as a finished workable technique for general use without consideration of any of the above topics, and examining DLS on these topics has academic merit as well as being immediately useful to practitioners.

The aims of this research were to discover any inadequacies that DLS may suffer - and hopefully to suggest some corrections - and to produce a tutorial guide on how to correctly use DLS in areas where confusion may arise.

The results which are dispersed throughout the sections, and listed in section 1.3.0, have both academic and non-academic applicability and I intend to publish them.

1.1.2 Semantic content of the ER model.

There are many different versions of Chen's basic ER model [Chen 1976]. Many of then are just additions to the basic model that helps represent higher level semantic constructs [Jajodia et al. 1984]. These semantic extras are used to help model situations which the basic E-R model has trouble representing correctly.

These higher level semantic constructs have been left out of DLS in an attempt to keep the technique simple. This is crucial to the philosophy of the technique which is that everybody should be able to master it quickly and easily regardless of their experience in the field of database design.
A major question to be addressed is how many of the advantages gained through having diagramming constructs that represent semantic concepts in record-based models are lost due to their omission in DLS.

This question can be split into 2 parts:

- does DLS produce the correct finished representations for these problems, and
- how much of the original semantic information can be obtained by examining the finished representations.

Section 1.2 The Research results.

1.2.1 Introductory section.

Each issue that is raised by DLS is dealt with in a section of its own. There is a lot of interaction between the sections as one issue raises points that are applicable to other sections.

Section 1.2.2 Attributes

1.2.2.1 Introduction

In the original definition of the E-R model, [Chen 1976], three basic semantic constructs are defined: entities, relationships, and attributes. An entity is, (informally), defined as a "thing which can be distinctively identified." Similarly a relationship is "an association between entities", and attributes are "descriptive information about entities" [Codd 1971]. Thus we see that an entity or relationship has different attributes that provides descriptive information. There is the restriction that an attribute must be an atomic value, because it represents a characteristic of an entity and therefore has no meaning of its own

These sorts of semantic constructs are typical of such record-based models. In opposition to this idea, fact-based systems dispense with entities altogether, (see section 3.3 of the Technical Report), and represents relationships and attributes as dependency information.

However, in the initial design stage of Kent's fact-based model [Kent 1984] there is a distinction made between facts about relationships and facts about attributes. The distinction is not emphasized in the later stages of the technique, but it is worth noting that some distinction is made. No distinction is made in DLS, as facts about both attributes and relationships are wholly grouped into SVDs and MVDs. An DLS MVD can only represent relationships, while SVDs can represent both attributes and relationships.

It is proposed that there are significant advantages to be gained from introducing to the dependency investigation stage of DLS a distinction between SVDs that represent attributes or relationships, as is done in the fact-based model.

1.2.2.2 Original E-R definition of an attribute.

"The information about an entity or a relationship is obtained by observations or measurements, and is expressed by a set of attribute-value pairs. "3", "red", "Peter", "Johnson" are values. Values are classified into different "value sets", such as "Feet", "Colour", "first-name", and "Last-name". An attribute can be defined as a function that maps from an entity or relationship to a value set or a cartesian product of value sets." [Chen 1976]

The important part of this definition is the "value sets". In the relational sense these are the "domains" which columns of tables will draw their actual values. Both attribute and relationship type facts will map onto a domain, (i.e. have a relevant value set), but the relationship type facts value sets will often be of a different nature that an attribute types. Attributes must be the rawest form of data possible, things which can have no purpose other than providing empirical or descriptive information. A relationship's value set will often be a foreign key to elsewhere, and it is possible to store further information about it.

1.2.2.3 Proposal.

The identification of attribute type facts is a useful concept because these can be made a lesser priority in the design stage than relationship type facts.

To aid in the tractability of designing large systems, it is recommended that identified facts about attributes be relegated to the final stages of the design.

Introducing this distinction separates DLS into 4 separate processes:

- identifying of the necessary fact information to be stored, and
- separating out attribute facts, and
- designing a skeleton structure of relationship facts, and
- fleshing out of the skeleton with the attribute facts.

This is a guide to the use of the DLS technique. It is an addition and not an alteration, (though some changes will be proposed later). The advantages of this proposal, with an appropriate example, will now be discussed.

1.2.2.4 Discussion.

The separating of relationship and attribute information is a process that a good database designer will do automatically. A tutorial on this topic would have been a valuable addition to Smith's original document.

The exact distinction of whether a fact is an attribute or relationship is not rigorous, and it need not be. The purpose of the distinction is to aid in the design of a database. No structural differences will be evident in the finished product if there is no clear distinction in a particular case. This is purely an assistance in the ordering of operations, that will ease the process of designing a large system. Heuristics are needed to reduce complexity because formal schema design is littered with NP complete problems, as stated in section i.

It is still possible for the designer to decide whether he/she will want to divide the schema into sectors - to complete one at a time - and then to merge them into one larger structure, or to approach the schema as a whole. The difference is that the first choice will be an iterative process of the four tasks, while the second completes each process sequentially. This choice is up to the individual, and the flexibility it provides is one of the advantages of the DLS technique.

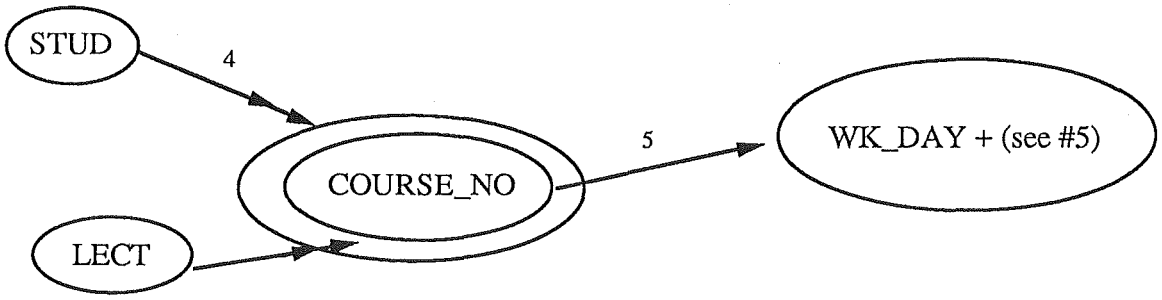
To illustrate this refinement, let us consider this scenario.

Dependency list.

1 -Every STUD student has a NAME and AGE.
2 -Every LECT lecturer has a NAME and RANK.
3 -Every COURSE_NO course has a TITLE and CREDIT_POINT.
4 -Every STUD takes several COURSE_NO courses.
5 -A COURSE_NO is taken by a LECT on a WK_DAY at a TIME_OF_DAY at a LOCATION class-room. A LECT may take several COURSE_NO.

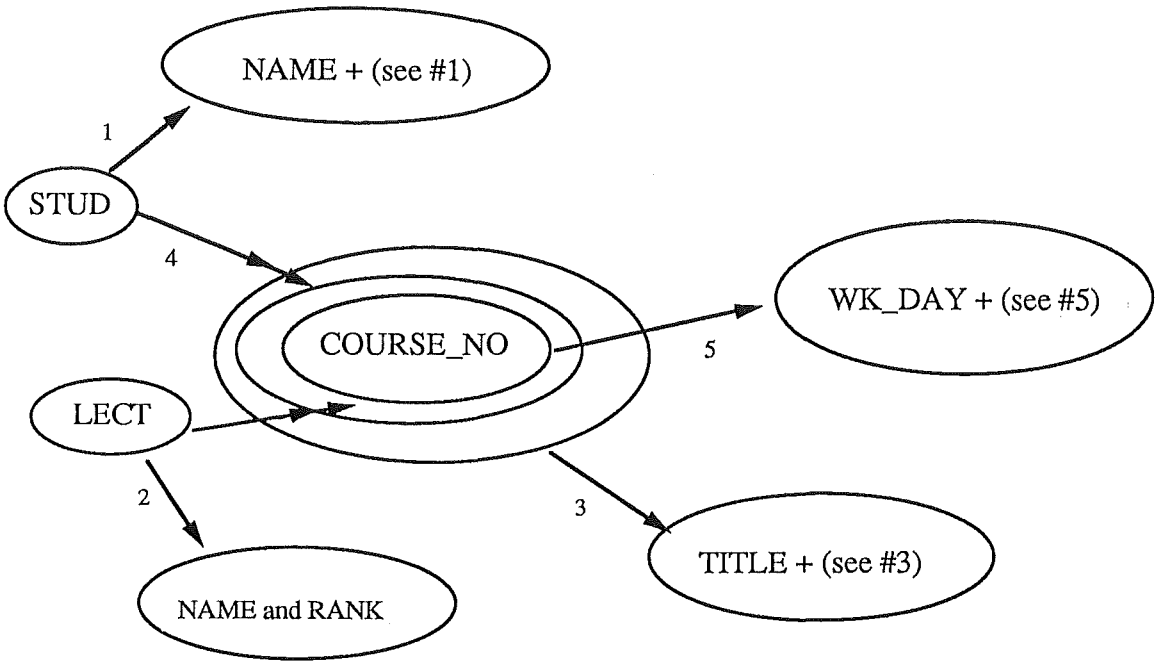
It is apparent that statements 1 through 3 are purely attribute information, while statements 4 and 5 contain facts that links the structure together. It is therefore recommended that statements 4 and 5 are diagrammed first:

Dependency diagram



The attribute information, (statements 1 - 3), can then be added to this skeleton structure to complete the diagram.

Dependency diagram



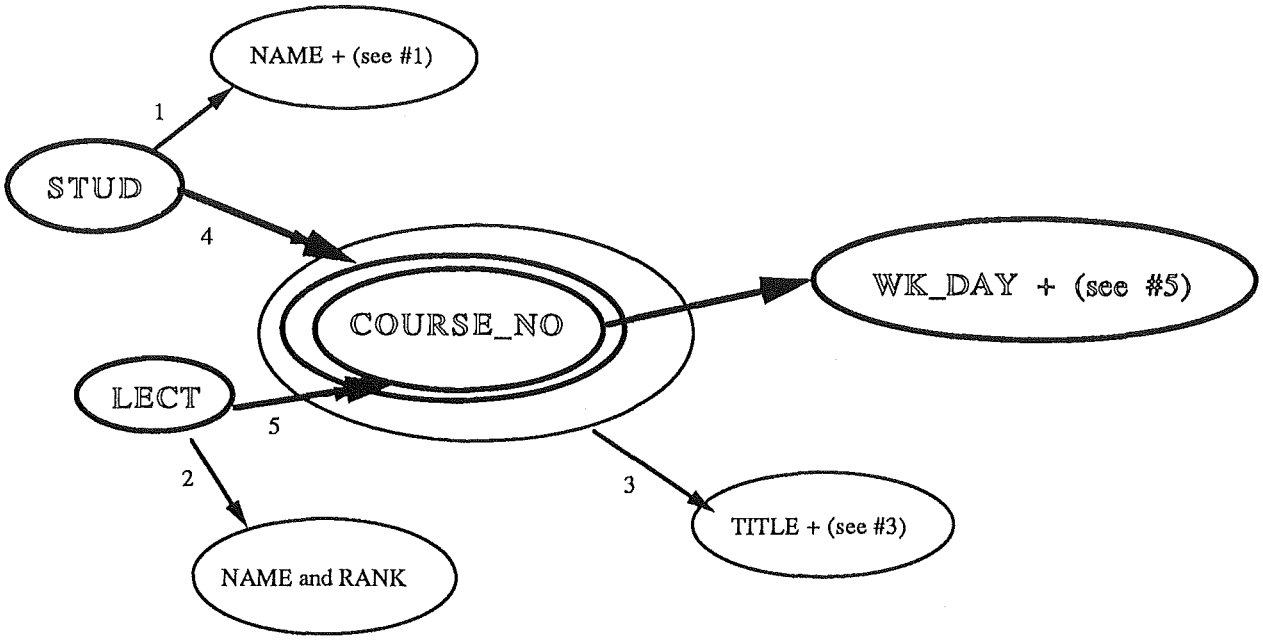
When the scale of the problem is much larger this is a useful heuristic.

The option of redefining SVDs so that distinction be made between attributes and relationships was considered in great length. The decision not to do so was primarily made because of two reasons:

- the problem of providing rigorous definitions of attributes and relationships that could be used to distinguish the two in difficult cases, and
- the fact-based school, with which DLS parallels, does not make any solid distinction either.

However, it is noted that highlighting the relationship type dependencies, and thus illustrating the "skeleton" of the schema does add useful information to a diagram. For example consider the difference that highlighting the more important relationship facts makes when compared with diagram above.

Dependency diagram



Producing diagrams of this type as documentation would be extremely useful, and they are very simple for a designer to implement.

1.2.2.5 Future applications - Use of the above.

It is predicted that an automated cartographic tool for the dependency diagramming and finished table stages of DLS will be produced. The scope for such a project is quite wide, and a specific recommendation would be the production of diagrams of the type of the figure above. Attribute dependencies may be dimmed, or temporarily removed, by such a tool so as to minimise complexity. This will help keep the number of facts that a designer must keep in mind within the "magic number" of seven plus or minus two. The magic number is the limit of number/facts/processes that the human mind can concentrate upon concurrently.

Such a tool could ease the process of database design in such a way as Computer Aided Software Engineering tools have aided systems analysis.

Section 1.2.3 Multivalued dependencies.

1.2.3.1 Introduction

Dependency information is used in deriving 5NF tables in decomposition approaches, and for synthesising tables in fact-based approaches. The definition of dependency information is similar in both cases, though the usage is different. Traditionally there has been 2 major classes of dependency information: SVDs and MVDs. There are many other dependency types, but they will not be dealt with directly, only as they appear in the discussion of MVDs.

1.2.3.2 Definitions.

Smith defines an MVD as:

"There is a multivalued dependency from A to B, (A ->-> B), if at any point in time, a fact about A determines a set of facts about B.

While examining the decomposition approach Date, [Date 1986], defines MVDs as:

"Given a relation R with attributes A, B, and C, the multivalued dependence (MVD)

R.A ->-> R.B

holds in R if and only if the set of B-values matching a given (A-value,C-value) pair in R depends only on the A-value and is independent of the C-value"

(Note that this definition requires that there be at least three attributes.)

The original definition of MVDs was formulated by Fagin [Fagin 1977b] when introducing the 4NF, and can be regarded as a standard.

"The multivalued dependency X ->-> Y is said to hold for R(X,Y,Z) if Yzx depends only on X, that is, if Yxz = Yxz` for each X,Y,Z` such that Yxz and Yxz` are non-empty."

The notation that is used is:

- R(X,Y,Z) - A table with three attributes.
- Yxz - All of the tuples of R that have a specific value for Y, and any value for X and Z.

These definitions are different was of defining the same term in increasing levels of formalisms. But just how comparable is the DLS definition compared to the others, and why is DLS the only definition that applies for a relation with only 2 attributes? (Note that for the remainder of this section the DLS definition of an MVD will be referred to as "DLS MVDs", and the other definitions will be referred to as "formal MVDs").

1.2.3.3 Discussion.

The usual definitions of MVDs are "tuple generating". This means that given a relation with an MVD and certain tuples, the existence of certain other tuples is assured, because of the MVD. Also formal MVDs satisfy a host of axioms and dependency rules.

1.2.3.3.1 Example cases of MVDs.

To illustrate the differences between DLS's MVDs and the formal definitions of MVDs a few examples will be considered.

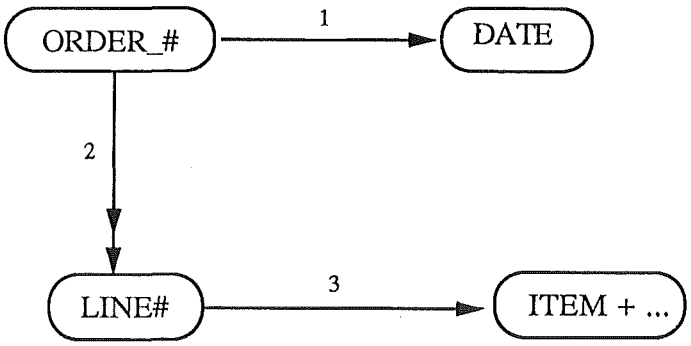
1.2.3.3.2 Example 1.

Consider this DLS representation of this scenario

Dependency list

1 -An ORDER_# of different invoices will be made on a DATE.
2 -An ORDER_# will have many associated LINE_# line numbers that stores the details of each line of the invoice.
3 -Each LINE_# has associated details of ITEM items that the order is of, and ....., (other SVD attributes).

Dependency diagram



Finished tables

ORDER_#	LINE_#	ITEM .....

Given this particular state of the table

INVOICE

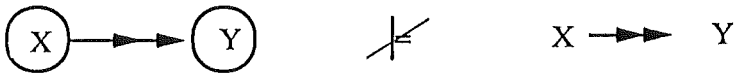
ORDER_#	LINE_#	ITEM .....
1	1	washer
1	2	screw
2	1	washer
2	2	screw

There is a DLS MVD between ORDER_# and LINE_# in the table INVOICE. For a formal MVD to exist there must also be these tuples in the table, (thus "tuple generating").

1	2	washer
1	1	screw

These tuples are clearly not desirable in the above table scenario, as they do not represent any natural state of the database.

It is therefore can be concluded that



That is if a DLS MVD holds between X and Y; X ->-> Y does not necessarily hold in the formal sense.

1.2.3.3.3 Example 2

To further illustrate the differences between the two definitions, consider this case where a formal MVD will hold, and the tuple generating property will make sense.

There is a relation with attributes (Room, Lecturer, Course), where a Room has Lecturers lecturing Courses in it. And there are 2 MVDs in this relation:

Room ->-> Lecturer, and
Room ->-> Course,

and not a ternary relationship between the three.

If there is a sample tabulation of this relation as

ROOM	LECTURER	COURSE
s1	Smith	ACCY 101
s1	Smith	RECR 303
s1	Brown	ACCY 101

To preserve the MVD information that "RECR 303" is taken in room "s1", and to keep it independent of the lecturer, there **must** be this following tuple in the table as well.

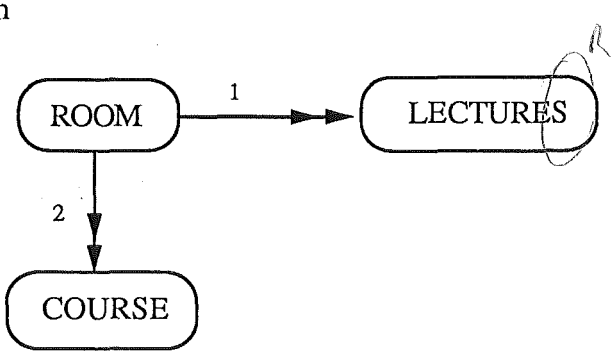
s1	Brown	REC 303

DLS will model the above case differently. Starting from the dependency list.

Dependency list

1 - A ROOM will have many LECTURES who will do things in it.
2 - A ROOM will have many COURSES taken in it.

Dependency diagram



Finished tables

ROOM	LECTURER

ROOM	COURSE

Notice that the correct MVDs are derived from the dependency list.

This can be explained by the differences between the methods that use the relevant definitions.

1.2.3.3.4 The differences between the approaches that use MVD definitions.

The formal definition of the MVD is used for decomposition approaches. The MVD information is used to decompose the Universal Relation into normalised tables. DLS is a synthesis method, which does not decompose from larger tables, but builds tables from the dependency information. Because the usage of dependency information between the methods are different, the format that dependency information is stored can differ as well.

In a decomposition technique the starting conditions of Example 2 would occur, i.e. the relation (ROOM,LECTURER,COURSE), as part of the decomposition from the Universal Relation. The formal MVD information be used to decompose this table into two new fully normalised tables. This step is usually done to achieve 4NF.

ROOM	LECTURER

ROOM	COURSE

The dependency information in the DLS case is used to derive the same tables without the intervening (ROOM,LECTURER,COURSE) relation. Thus if synthesis techniques are used properly relations that are tuple generating should be avoided. This is because no relation with 2 MVDs would be properly synthesised.

MVDs in synthesis are therefore only used to derive the keys of tables, as opposed to being used to derive new relations from existing ones.

Synthesis methods have been called "binary relation" methods, (see section 3.3 in the Technical Report), because they only deal with dependencies between two attributes at a time. The case of how a formal MVD with only 2 attributes has yet to be examined.

1.2.3.4 MVDs in relations with only 2 attributes.

It is intuitive that an MVD can exist in a relation with only 2 attributes, but it is a special case. Fagin devotes a section to this special case in [Fagin 1977b]. Wishing to avoid duplicating the complicated mathematical formalisms that is used we will state these conclusion:

- an MVD may exist in a binary relation R(A,B), except
- if R(A,B) is by definition the cartesian product of A and B, then there is the formal MVD 0 ->-> A, (or equivalently 0 ->-> B), and not A ->-> B.
- In this special case, to preserve 4NF, this relation must be decomposed into two separate relation R1(A), and R2(B).

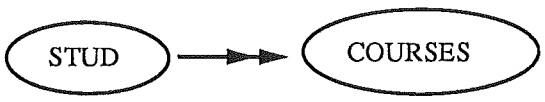
These conclusions may be used to examine DLS performance, considering its own definition of an MVD, when dealing with binary relations.

Consider this scenario:

Dependency list

1 -Every STUD student takes every COURSES honours paper that is offered by the COSC university department.

Dependency diagram



Finished table

STUDS	COURSES

But as a student takes all of the papers we note that 0 ->-> STUD, i.e. the empty set defines STUD, and the resulting tables are the cartesian product of STUD and COURSES. By 4NF guide-lines this should be decomposed into two separate projections:

Dependency diagram

(Note by strictly using the Smith's definition of an MVD the initial dependency diagram, (with the MVD), is correct.)

1.2.3.5 Conclusions.

By examining the usage of MVDs we have been able to establish that though the DLS definition of an MVD is substantially different to the formal definition it provides the necessary information to perform its desired task. The information that a DLS key provides will hopefully be used to synthesise 5NF tables, by indicating the correct key structure for a relation. This task is examined in section 1.2.6.

The illustrated problem with the incorrect identification of cartesian products is not a major inadequacy but it is worth noting. A potential user should be aware of the problem, and that such problems do exist. Therefore further investigations of this type will be discussed.

Section 1.2.4 Semantic Constructs.

1.2.4.1 Introduction.

In this section we address the question of how well DLS represents semantic information, and whether it can model different semantic constructs correctly.

One of the major advantages of record-based models is the ease in which high level semantic information can be represented directly within the model. A case in point is the EER model of Teory, Yang, and Fry [Teory et al. 1986]. In this technique, diagramming structures are given to each individual semantic detail. A comparison will be performed with these structures and the corresponding DLS diagrams.

1.2.4.2 Hierarchy definition.

Two important classes of semantic construct will be dealt with directly: subset hierarchies, and generalisation hierarchies. These concepts are abstraction concepts. "View integration, for example, requires the use of abstraction concepts such as generalisation" [Navathe et al. 1986].

To define these abstraction concepts, entities must be reintroduced.

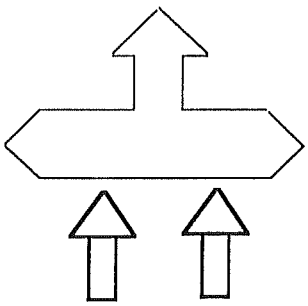
"Subset Hierarchy Definition. An entity E1 is a subset of another entity E2 if every occurrence of E1 is also an occurrence of E2."

"Generalisation Hierarchy Definition: An entity E is generalisation of entities E1, E2, ..., En if each occurrence of E is also an occurrence of one and only one of the entities E1, E2, ..., En."

The EER model specifies special diagrammatic representation for each concept.

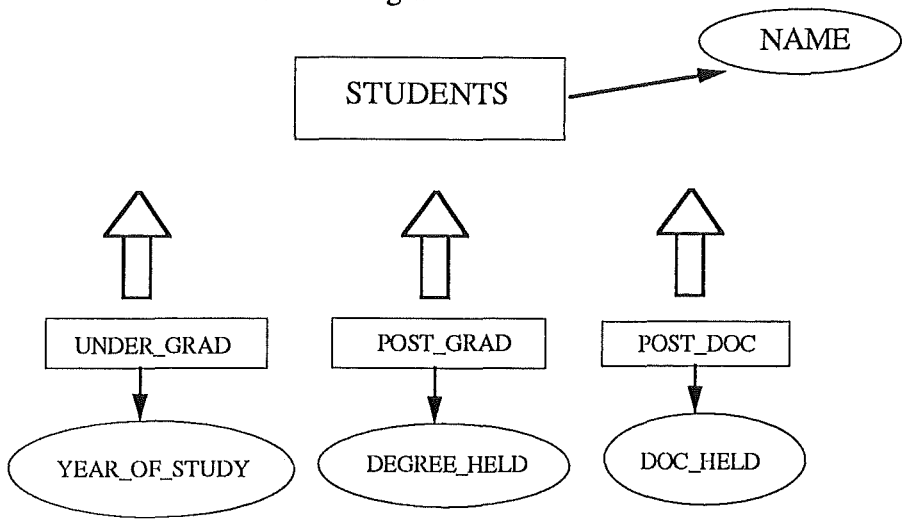
Subset Hierarchy

Generalisation Hierarchy



We wish to examine a specific example of subset hierarchies and see what DLS will produce.

Consider this EER diagram



(Note that the members of the subset are mutually exclusive.)

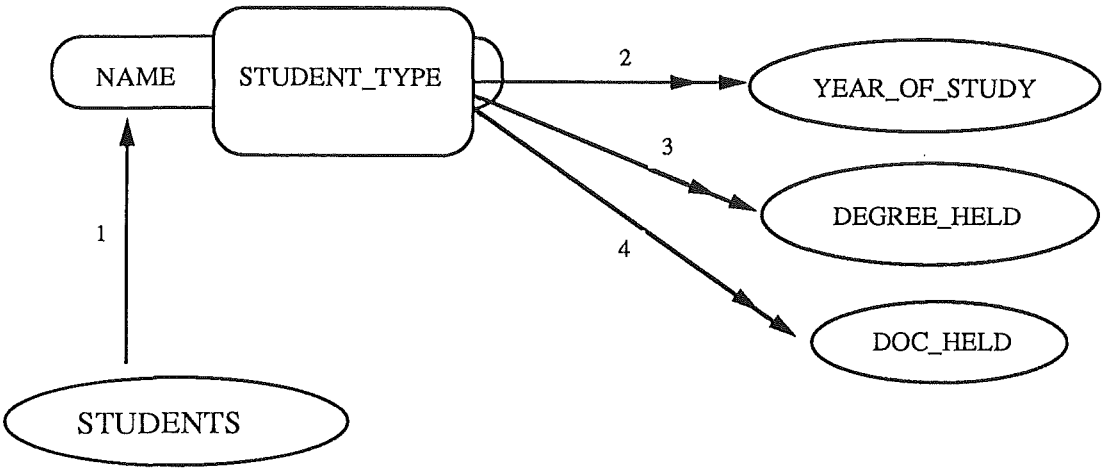
1.2.4.3 Discussion.

To represent this subset there may be a distinct identifier for each of the three alternatives. Though this is a possibility, in this case it is unlikely. However it is desirable to separate the three attributes YEAR_OF_STUDY, DEGREE_HELD, and DOC_DEGREE from each other. What would appear to be the best solution is to introduce an attribute of STUDENTS called LEVEL_OF_STUDY, which can have one of the three relevant values. This is introducing a flag, which is what is generally done in the producing tables step of the EER approach. Applying this example to DLS will result in this representation:

Dependency list

1 - STUDENTS have NAMES, and are either an undergraduate, postgraduate, or postdoctorate STUDENT_TYPE.
2 - STUDENT_TYPE Undergraduates are in a YEAR_OF_STUDY.
3 - STUDENT_TYPE Postgraduates already have DEGREES_HELD.
4 - STUDENT_TYPE Postdoctorates already have DOC_DEGREES.

Dependency diagram



Finished table

STUDENTS	STUDENT_TYPE	NAME

STUDENT_TYPE	YEAR_OF_STUDY

STUDENT_TYPE	DEGREE_HELD

STUDENT_TYPE	DOC_HELD

The finished table that would be produced by applying decomposition rules to the EER diagram of above are exactly the same. Therefore we see that stating the dependencies between the attributes has produced suitable finished tables.

1.2.4.4 Conclusions.

The fact-based model makes provisions for such concepts as aggregation, and the correct finished tables should result. The problem is that semantic information is not documented anywhere, and is invariably lost. Significantly, by using dependency lists, DLS provides a free reign to the designer to list any semantic information that he/she see fit. This aids in the retention of necessary semantic information. (Note that by stating the dependencies clearly, most semantic information appears as a by-product, as is evident in dependency list statement 2.)

We conclude that the fact-based model will lose semantic information unless a further documentary stage is introduced. By introducing dependency lists DLS has increased how much semantic can be represented.

Section 1.2.5 Fan-Traps.

1.2.5.1 Introduction.

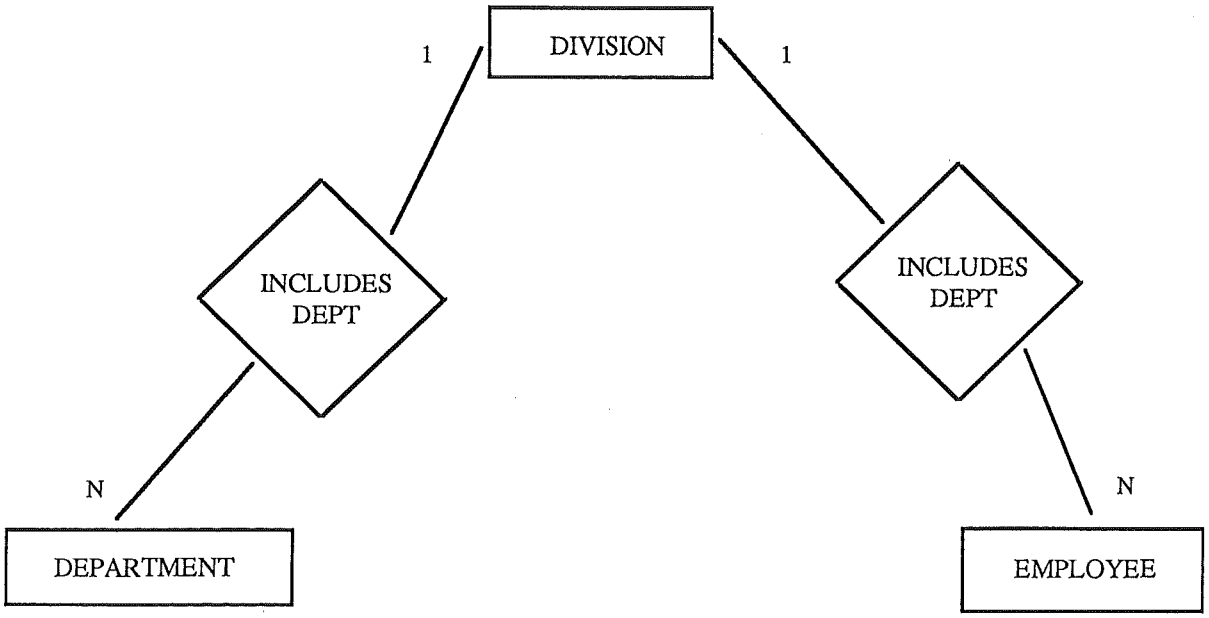
The problems of fan-traps is treated in [Howe 1983], and [Bowers 1988]. The essence of the problem is that the incorrect linking up of entities can lose information that would otherwise be available in an alternative structure.

1.2.5.2 Ordinary Fan-traps.

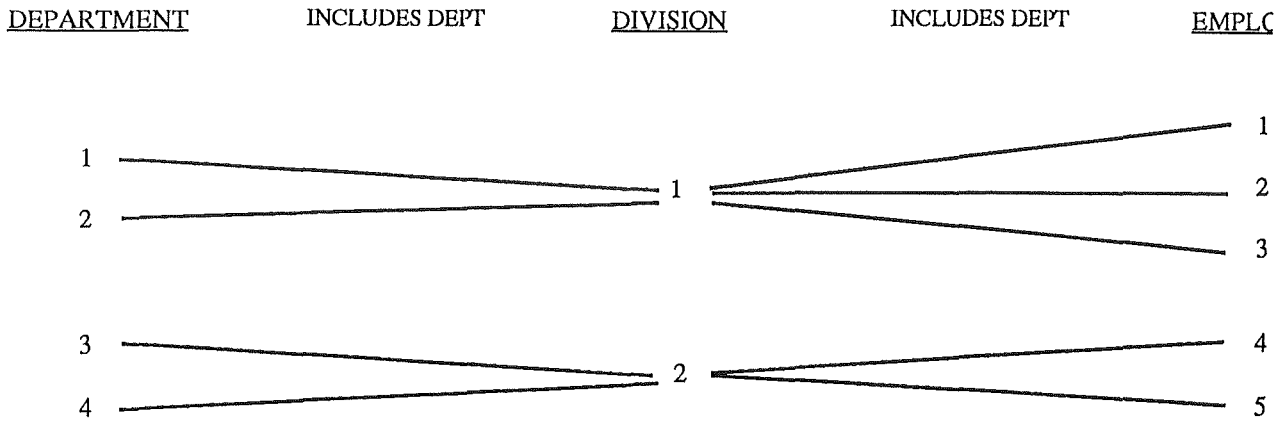
One type of a fan-trap is the ordinary fan-trap.

1.2.5.2.1 Description.

Consider this E-R representation:



The problem with this structure can be illustrated as:



This fan shape is losing the connection between an employee and a department. Specifically it is impossible to ascertain which department an employee belongs to.

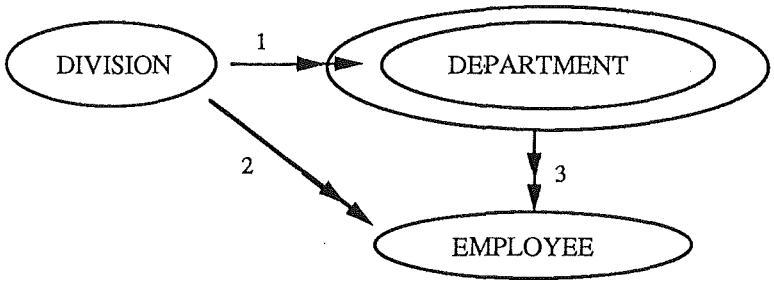
1.2.5.2.2 Discussion.

This scenario can be represented in DLS as:

Dependency list.

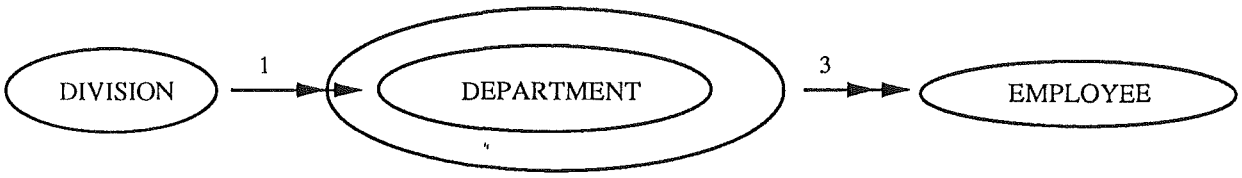
1 - Each DIVISION has many DEPT departments.
2 -Each DIVISION has many EMPLOYEE employees.
3 -An EMPLOYEE will belong to a department, (i.e. a department will have many employees).

Dependency diagram



But in this case the writing of the dependency list, particularly the last sentence, has helped in identifying a data relationships that otherwise may be missed by producing a transitive dependency. In this sense the method is helping the designer to be organised and has helped in the prevention of fan-traps. Therefore statement 2 will be removed from the dependency list.

Correct diagram



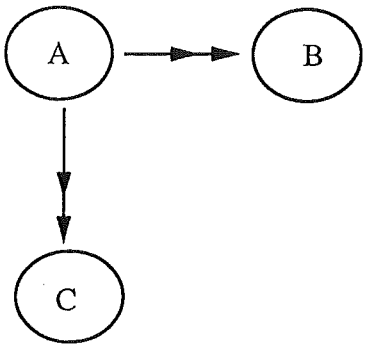
Finished table

DIVISION	DEPARTMENT

DEPARTMENT	EMPLOYEE

1.2.5.2.3 Recommendations for this case.

A particular case to be wary of is when a bubble has two MVDs coming out of it. As well as being possible fan-traps, the organisation of MVDs may also not satisfy 4NF, (see section 1.2.6.6).



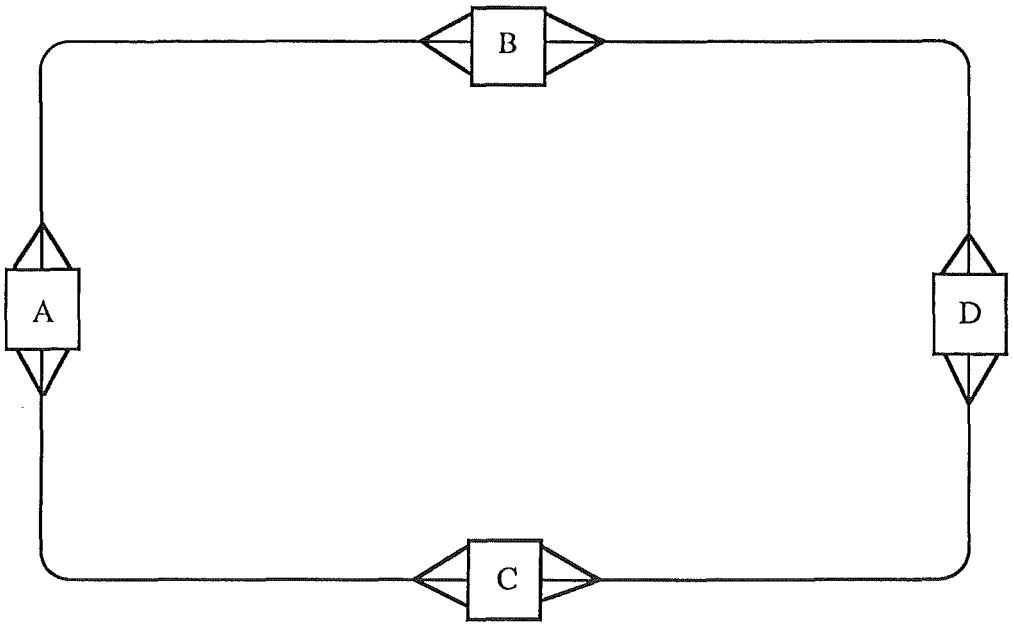
It is recommended that an examination of what possible relationships exist-between B and C-be carried out. There is a high possibility that there may be a potential fan-trap, or that A, B, and C are involved in a ternary relationship-which will be examined later. Another issue that is raised is the possibility that an employee does not

actually belong to a division at all. This is the optional/mandatory issue and will also be considered later.

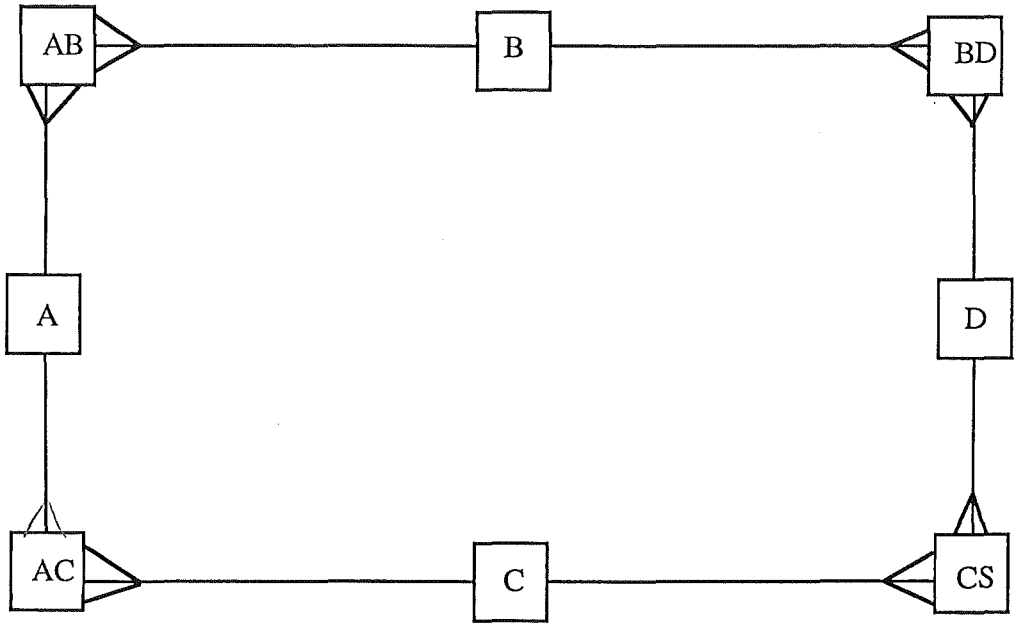
1.2.5.3 Complex Fan-traps.

The complex fan-trap is introduced in [Bower 1988]. It is illustrated in the E-R notation, so the initial description here of the problem will also use E-R diagrams. It is a particularly nasty combination of "many:many" relationships.

A complex fan-trap is illustrated by the following diagram.

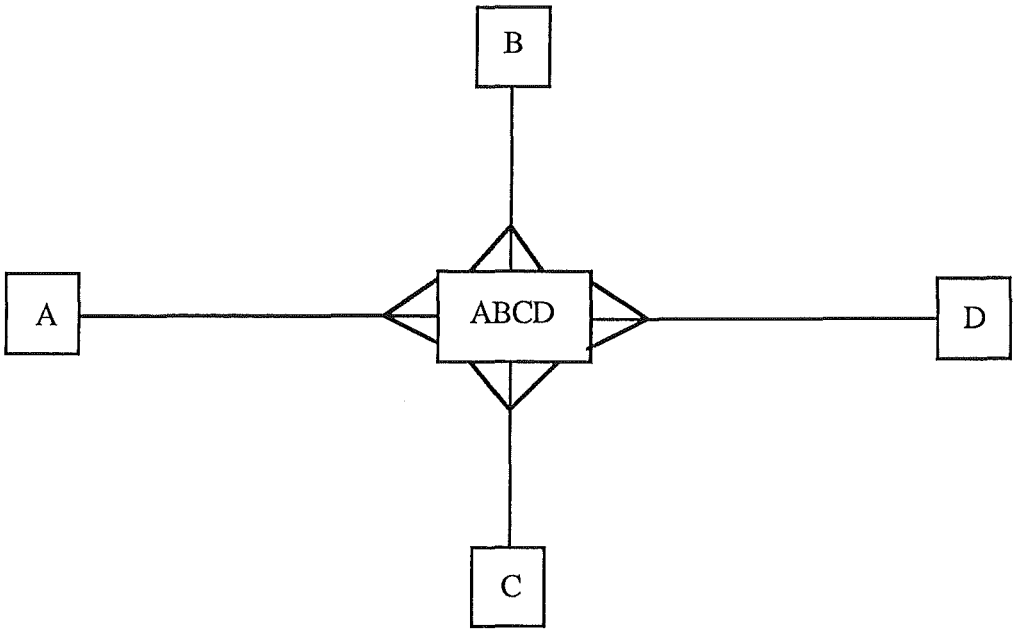


When following the decomposition guidelines of relationship relations, this diagram can be converted to



Where the new corner entities are "relationship entities". Now their are 4 separate fan-traps, one for each relationship entity. The problems are the same as the ordinary case, except that they are now compounded. This can be illustrated by considering how to get any relevant information about the B entity, given a particular C value.

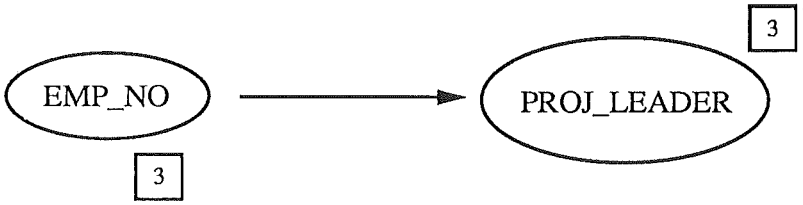
The correct way to deal with this problem is instead of creating four relationship entities, only one should be created but it should have all four entities as the key.



It can be concluded that when a complex fan-trap occurs, the designer has failed to correctly identify a n-ary relationship [Bower 1988]. N-ary relationships in DLS are dealt extensively in section 1.2.8, and it will be shown that a cyclic structure will result if DLS incorrectly models the above situation. Guidelines on how to correct this problem are also provided.

1.2.5.4 Conclusions

It is concluded that all semantic information must be stored in the dependency list. The purpose of the dependency diagram is to map the dependency information onto the finished tables. As the only real semantic information that a finished table stores is the dependence of the attributes on the primary key, the dependency diagrams only task is to represent the dependency information, (note that the foreign key information is included in this stage because the finished tables also store this information).



All semantic information must therefore be carefully worded in the dependency list statements. However, clearly stating the dependencies between data fields clearly will often illustrate semantic information without any extra effort.

Section 1.2.6 Normal forms.

1.2.6.1 Introduction.

Smith makes the claim that

"tables satisfying each normal form (5) can be composed directly from these diagrams".

Examples for each normal form, that are based on [Kent 1983], are then illustrated. The claim that the tables produced by DLS will automatically be in 5NF is something that should be considered with scepticism. Even the most formal synthesis algorithms can only achieve 3NF or 4NF in special cases. Why should Smith be able to claim 5NF? Adding to this scepticism was that in the normal form demonstrations, absolutely no mention of Boyce/Codd normal form (BCNF) was made at all!

It was therefore decided that some tests of a particularly difficult nature, each designed to test a particular normal form, be performed. The given definition for each normal form is from Date.

1.2.6.2 1NF.

A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only.

First normal form is a requirement for the use of the technique. It should therefore not be regarded as a result, but as a starting point. Of course the procedure will preserve 1NF throughout.

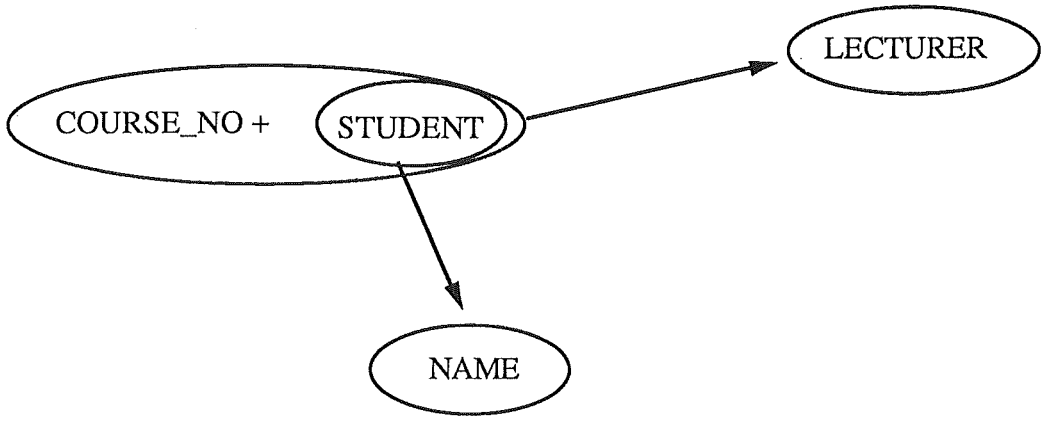
1.2.6.3 2NF.

A relation R is in second normal form (2NF) if and only if it is in 1NF and every non key attribute is fully dependent on the primary key.

DLS has no trouble producing tables that are 2NF. Fact-based, and synthesis techniques traditionally have been able to produce 3NF diagrams due to their dependency oriented nature.

Posit that a STUDENT, with a NAME, takes COURSES lectured by a LECTURER. One table with these four data fields, with a composite primary key of STUDENT and COURSE, is not 2NF because a students NAME depends only upon the STUDENT his/herself. These facts are shown below on a dependency diagram from which the necessary projected tables have been composed.

Dependency diagram



Finished Table

COURSE_NO	STUDENT	LECTURER

STUDENT	NAME

These tables are 2NF, (though they are not in a higher normal form as shall later be demonstrated).

1.2.6.3 3NF

A relation R is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

1.2.6.4 Discussion

DLS has no trouble producing this normal form. The given example in the original paper is sufficient enough to represent this.

Consider the following facts

Dependency list

1 - an EMPLOYEE works in one DEPT department,
2 - a DEPT has one LOCATION

If one table is created

Finished Table

EMPLOYEE	DEPARTMENT	LOCATION

that table is not 3NF, since LOCATION is determined by DEPT. The dependency diagram below represents these facts as

Dependency diagram

and the derived 3NF tables from the diagram are

EMPLOYEE	DEPARTMENT

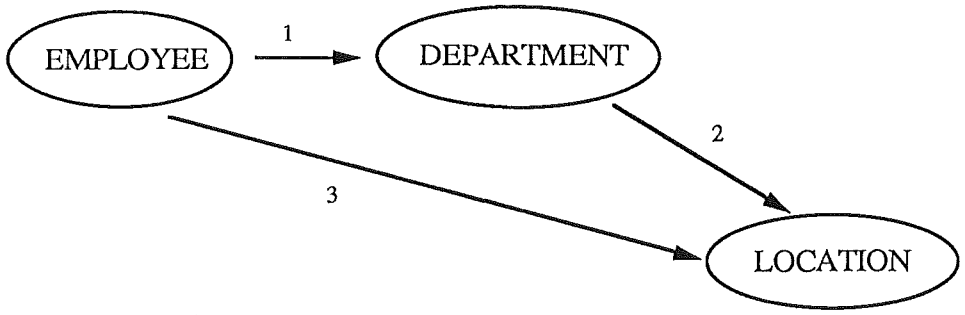
DEPARTMENT	LOCATION

These diagrams are the final representation of the two original dependency statements.

1.2.6.4.2 Transitive Dependence

If a third fact - an EMPLOYEE works in one LOCATION - were to be added to the two existing facts, then the dependency diagram would appear as

Dependency diagram



The new fact is a transitive dependence. This case is specifically warned against in Smith's paper, and this new dependency, (number 3), must be erased.

So far there have been no problems in attaining 3NF tables, as was to be expected.

1.2.6.5 BCNF

1.2.6.5.1 Introduction.

A relation R is in Boyce/Codd normal (BCNF) form if and only if every determinant is a candidate key.

This normal form was attacked with special interest due to its complete omission in the original paper. Every normal form discussed so far has exclusively dealt with SVDs, while 4NF is the first to introduce the MVD. BCNF is a sort of bridge between the two, and is the highest level of "normality" necessary for relations that only have SVDs, (higher normal forms are implied in this case).

1.2.6.5.2 DLS performance w.r.t BCNF.

To illustrate the problem of BCNF consider a development of the STUDENT, LECTURER example that was used in the 2NF section.

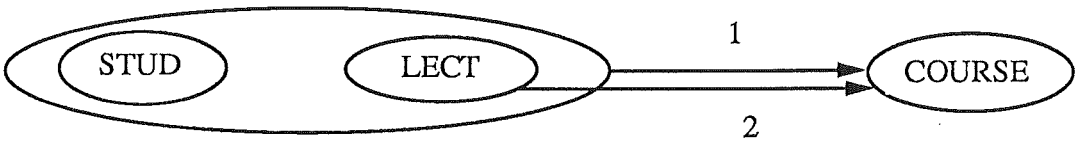
Dependency list

1 - Each COUSE is taught to a STUD by one LECT lecturer.
2 - Each LECT lectures only one COURSE. Each COURSE is taught by several LECT's.

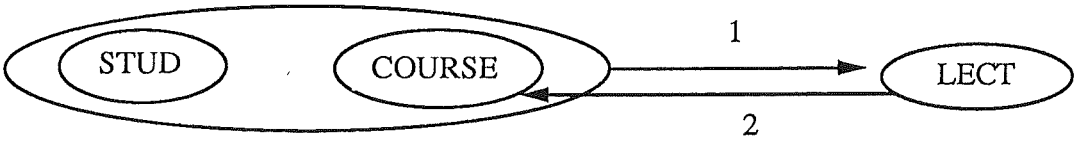
Any purely attribute information that may exist in this model - like students' names - are ignored as they only cloud the issue. The second sentence in statement 2 is there to specifically illustrate that there is not a dependency of LECT on COURSE. This dependency list could be initially diagrammed as one of three alternatives

Dependency diagram

or



or



The advisability of the first alternative is questionable, while the second alternative holds obvious redundancies, i.e. the fact that a LECT lectures a COURSE is stored twice. Therefore the third alternative will be considered to be the most suitable

representation of the two facts as stated above. This diagram is used to derive the following table.

Finished table

T1

STUD	COURSE	LECT

T2

LECT	COURSE

This is a perfectly legitimate result in the Smith's technique as it stands! An example tabulation of these tables follows.

T1

STUD	COURSE	LECT
Fred	COSC	Bedrock
Fred	ROCK	Kermit
Wilma	COSC	Bedrock
Wilma	ROCK	Piggy

T2

LECT	COURSE
Bedrock	COSC
Kermit	ROCK
Piggy	ROCK

We notice that these relations contain redundancy, because the LECT-COURSE fact is stored in separate places, (Note that in T1 when given a certain LECT there is only one possible corresponding COURSE). However, it is not valid to erase LECT from T1, (to make it an all key relation), because a students LECT is not derivable from COURSE alone, (because more than one LECT lectures a given COURSE). The problem is that T1 is in 3NF but it is not in BCNF. The trouble is that LECT is a determinant, i.e. it derives one of the other attributes in the relation, but it is not a candidate key of the relation.

Decomposition dictates that relations of type T1 be decomposed into several different relations. The recommended decomposition [Date 1986] requires that two BCNF projections

T3

STUD	LECT

T4

LECT	COURSE

be produced. Notice that T2 and T4 are exactly the same. Somehow the DLS technique has failed to produce the correct BCNF tables.

1.2.6.5.3 Investigation into correcting problems.

At this stage let me placate any proponent of the DLS technique by stating that there is a correct way for the technique to represent this normal form, the problem is making sure that the user derives the correct result.

To derive the correct normal form structure, let us start with the correct finished tables and work backwards to derive the correct dependency diagram and statements.

Taking the finished tables as being the all key table T3 and the table T4 provides a starting point for this backtracking process. For table T3 to be all key there must be an MVD between STUD and LECT.

STUD ->-> LECT, or
LECT ->-> STUD.

Examining the original dependencies it is deducible that

STUD ->-> LECT

is the correct dependency.

The table T4 is a simple SVD between LECT and COURSE.

LECT ->-> COURSE.

It is deducible that LECT defines the values of COURSE because LECT is a key of T4 .

Notice that T3 and T4 are distinct tables, and therefore they must be distinct facts. The LECT participation in both facts must therefore be distinguished.

The dependency diagram that will derive these tables is

(Note that a doublebubble is used to distinguish the two distinct facts about LECT. If the two facts are not correctly distinguished, i.e. a doublebubble is not used, then the T1 table will recur.)

Taking this reversal procedure one further step backwards we can predict that the dependency list statements that would produce these diagrams are

Dependency list

1 - Every STUDENT will have several LECT.
2 - A LECT will only lecture ONE COURSE.

The first point that strikes when comparing this list with the original is that an MVD has been introduced.

A proponent of DLS would be justified in arguing "Well obviously the original dependency list is wrong, and the second one should have been used from the start".

Unfortunately I do not consider it to be as simple as that. One of the purposes of the technique is that somebody who does not know the formalisms of database design can use the technique to produce fully normalised tables. It is highly probable that a user who is not intimately familiar with normalisation theory will produce the first dependency list rather than the second. The reason for this mistake is that there is a logical dependency between a STUD and the COURSEs that they take.

Using the definition of a SVD the original statement 1 is perfectly legitimate, it is just that there is a better way of representing the dependencies in this case. This is the basis for the decomposition approach. Starting with a "Universal relation" with many

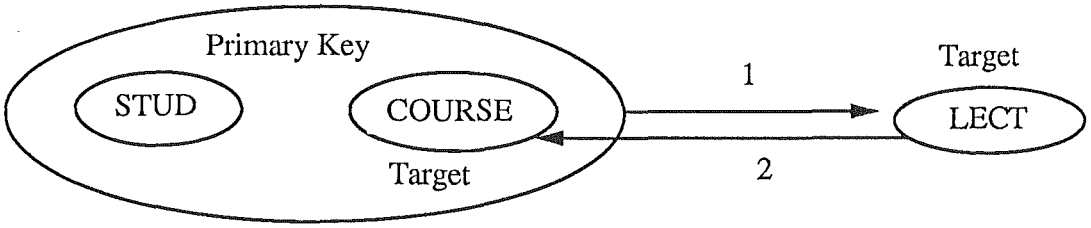
dependencies, smaller relations are derived, with dependencies that sometimes have no direct representation in the original "Universal relation" [Kent 1983].

My interpretation of Smith's proposal is that this decomposition step is by-passed and the correct normalised tables be produced directly through the use of "rigorous" dependencies. It has been shown, however, that these dependencies are not rigorous in all cases, and that some interpretations must be made to deduce which set of dependencies most correctly models the data.

This problem can be avoided by the identification of such problem areas, especially with BCNF, and the documenting of the correct decisions that should be made. This was not attempted in the original paper, and the author is not aware of any subsequent publication which addresses these issues. Publication of my results is thus likely to benefit users of DLS.

1.2.6.5.4 How to identify problem cases.

The problem that has specifically been identified occurs when a field of a prime key (COURSE) is itself a target of the target field (LECT) of that prime key.



The recommended action, in this case, is that the target field of the prime key be removed and the SVD be altered to a MVD. This alteration preserves BCNF.

This rule still holds if the prime key has more than two attributes.

1.2.6.6 4NF.

1.2.6.6.1 Introduction.

A relation R is in fourth normal form (4NF) if and only if, whenever there exists an MVD in F, say A ->-> B, then all attributes of R are also functionally dependent on A. In other words, the only dependencies (FDs or MVDs) in R are of the form K -> X (i.e., a functional dependency from a candidate key K to some other attribute X). Equivalently: R is in 4NF if it is in BCNF and all MVDs are in fact FDs.

1.2.6.6.2 Discussion.

Given that the additional guidelines for achieving BCNF are included in the DLS technique, all that is necessary to achieve 4NF is the correct identification of all of the MVDs. The examples used in the DLS paper will be augmented by another example to show that 4NF is achievable.

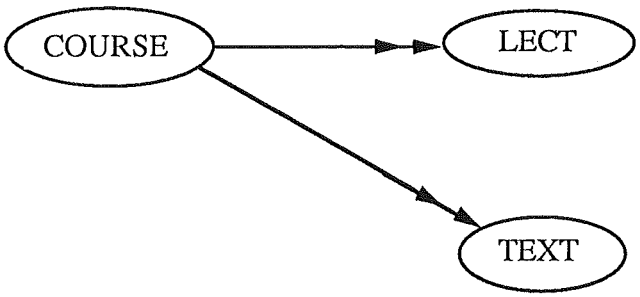
Alter the previous example so that these are the relevant dependency list statements:

Dependency list

1 - A COURSE is taken by many LECT.
2 - A COURSE may use many TEXT books. A LECT may use more than one TEXT for a COURSE.

The second sentence in statement 2 is specifically stating that there is not a SVD between LECT and TEXT.

Dependency diagram



Finished tables

COURSE	LECT

COURSE	TEXT

These are the desired tables. Avoiding a table that combines LECT and TEXT is the major concern at this stage, but as long as the MVDs are correctly identified then no problem will occur.

1.2.6.6.3 4NF as it relates to fan-traps.

This is illustrated by replacing the second statement with

2 - a LECT uses many TEXTS.

These dependencies are a glaring error because of the tables that result.

Dependency diagram

Finished tables

COURSE	LECT

LECT	TEXT

This has missed the dependency between TEXT and COURSE, and therefore the information of what COURSE a TEXT is used for is unattainable. This mistake is caused by not getting all of the information, and this problem is discussed as the fan-trap problem, (see section 1.2.5).

1.2.7.6 5NF.

1.2.7.6.1 Introduction.

A relation R is in fifth normal form (5NF)-also called projection/join normal form (PJ/NF)-if and only if every join dependency in R is a consequence of the candidate keys of R.

In practice this means that a relation may be non-loss decomposed into 3, (or more), relations but not into 2. Unfortunately, in decomposition approaches to formally show that the necessary Join dependencies exist in the presence of MVDs etc. is NP complete. [Date 1986]

1.2.6.7.2 5NF as it relates to n-dependency and dependency information

5NF is a rather rare occurrence. It is a sufficiently well defined condition that the only investigative example would be to rename all of the fields in the example in the DLS paper and illustrate that 5NF still holds. The example in the original paper is proof enough.

The question of whether a relation should be n-decomposed, (to use Date's term), to gain 5NF, or whether it is a n-ary relationship is a relevant issue. This will be addressed in section 1.2.8.

1.2.6.8 Conclusions of Normal form investigation.

In the "Additional Guide-lines" section of DLS paper there is

"Since facts can always be stated and diagrammed in either of two obverse ways-A to B or B to A-how should the designer proceed when developing a dependency list and diagram? Although the designer's intuition is usually correct, these guide-lines should help:"

Then several guide-lines are listed. It has been shown that to derive tables of 5NF it is absolutely necessary, indeed vital, that the correct order of dependency information be found, the "designers intuition" is not enough.

It is therefore proposed that the recommendations that were introduced at the end of sections 1.2.6.5.3, 1.2.6.5.4, and 1.2.6.6.3 be introduced as part of the DLS technique.

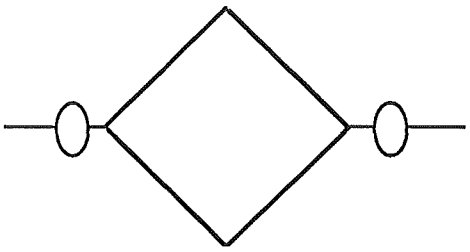
Section 1.2.7 Optionality, (or membership class).

1.2.7.1 Introduction.

The concept of a dependency being optional is not dealt with in great detail in the DLS paper. An example of dependencies that are optional is in Section 1.2.4.2. For a given student only one of the dependencies in statements 3 - 5 will exist.

Models like the EER model, and Martins Crows-feet diagrams [Martin et al. 1985] provide specific diagrammatic symbols for optionality.

EER



CROWS FEET

1.2.7.2 Discussion.

DLS can only represent the fact that a dependency may be optional by explicitly stating so in the dependency list. It is a curious omission in the diagrammatic phase of the technique.

The semantic information of optionality is especially important when the Data Manipulation Language applications are been developed. The error of assuming that a

particular tuple in a relation exists when it does not can be avoided by flagging that a dependency is optional.

A possible explanation for optionality not been included as part of the diagramming process is that the diagrams are "Dependency diagrams" only. Thus a diagram is only supposed to represent dependency information. However, domain information is also included in dependency diagrams, and optionality is much more an attribute of a dependency, than are the domains of the fields that these dependencies define.

However, the finished table representation will not have the optionality constraints, but it will represent domain information . If a table could exist, then there must be a corresponding table definition. Therefore as the finished table do not use this information there is no need for it to appear in the dependency diagrams.

In practice the problem of not flagging an attribute or relationship as been optional will appear in the application programs that use the database. A way of providing the necessary information so that the application writers can ascertain whether a relation or attribute will exist in a database is invaluable, but it is arguable that it is over and above the schema design process.

1.2.7.3 Conclusions.

It would be exceedingly useful to put the semantic information of optionality into the dependency diagrams. However, optionality is not necessary for producing finished tables, and one might argue that by adding this piece of semantic information, why not add another piece of information as well. The end result would be that it is no longer a fact-based technique but a record-based one and the diagrams would end up being E-R diagram clones. The philosophy of fact-based design prohibits this, because unnecessary complexity in the technique would be avoided.

Section 1.2.8 N-ary relationships.

1.2.8.1 Introduction.

Relationships between entities exist in different degrees. A relationship of degree one is called unary, a relationship of degree two binary, a relationship of degree

three ternary ... , and a relationship of degree n n-ary, (see Date [Date 1986], and Howe [Howe 1983]).

In this section we examine DLS's performance when confronted with relationships of different degrees. The DLS paper makes no mention of the concept of the degree of a relationship at all.

The analogous concept to the relationship degree in DLS is the number of uplink and prime keys that a target bubble may have. The degree concept is more traditionly associated with entity-based models. There are accepted finished table representations for relations of different degrees, and our investigation will take the form of stipulating a scenario where a known relationship of a certain degree exists, and testing if DLS will produce the correct tables.

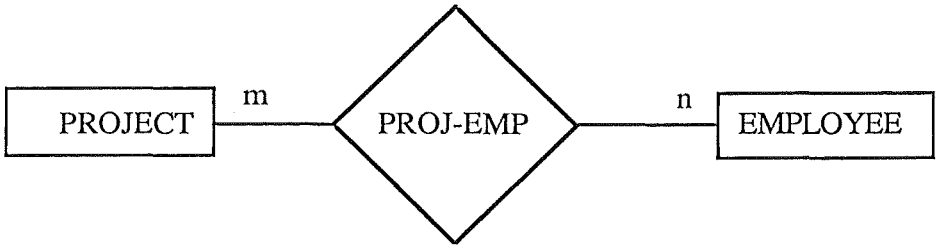
1.2.8.2 Binary relationships.

1.2.8.2.1 Introduction.

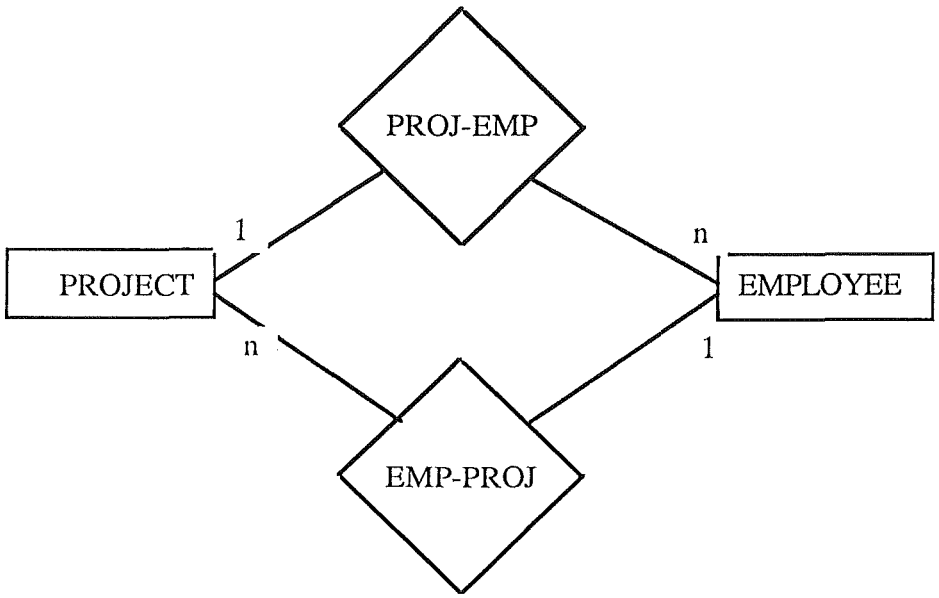
In record-based models there are three types of relationships that correspond to binary relationships: many:many, (many to many), many:one, and one:one. DLS has no trouble dealing with the latter two, it is the many:many relationships that are of interest. The E-R model will be used to illustrate an example of a many:many relationship, (adapted from Howe).

1.2.8.2.2 DLS performance w.r.t binary.

There exists a many:many relationship between PROJECT and EMPLOYEE such that an employee can be assigned to many projects, and a project may have many employees assigned to it.



This relationship can not be separated into 2 separate one:many relationships directly.



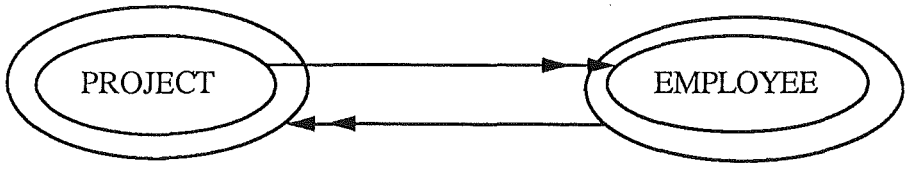
This is because in the formal E-R model the relationships PROJ-EMP and EMP-PROJ imply that formal MVDs exists between EMPLOYEE and PROJECT, and vice versa. This is is not the case, (see [Howe 1983] for further details).

Consider how DLS will handle such a problem:

Dependency list

1 - An EMPLOYEE will be assigned to many PROJECTS.
2 - A PROJECT will have many EMPLOYEES assigned to it.

Dependency diagram



(Note that the doublebubbles must be used because producing a table from a diagram that only used singlebubbles would be an impossible task.)

Finished tables

PROJECT	EMPLOYEE

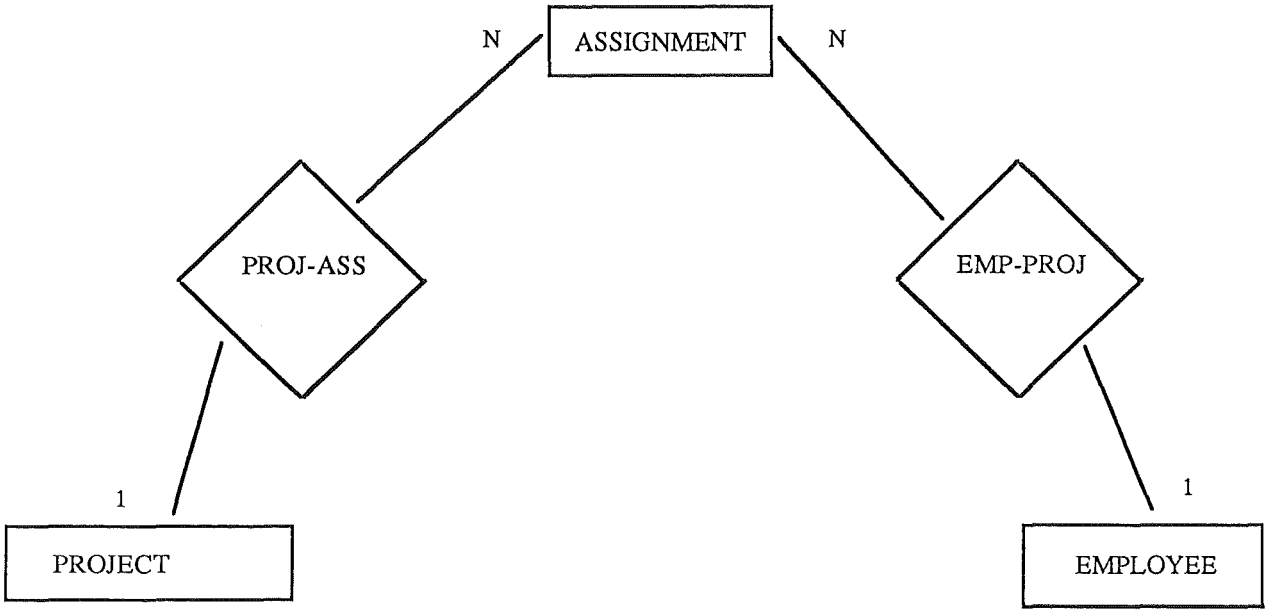
PROJECT	EMPLOYEE

This demonstrates that only one relationship between the two data fields exists! It is helpful to view one relationship as being the inverse of the other. These tables are obviously sub-optimal. Notice that the resulting structure is the correct representation of this dependency list and that it is a legal result.

1.2.8.2.3 Problems illustrated.

There is a far more desirable way of representing this scenario that eliminates this duplication of data.

The E-R model decomposition rules requires that many:many relationships be treated as a separate entity, and that a relation that has the participating entities as the key be produced from it. This decomposition has an E-R representation of



this will produce the tables

PROJECT	....

EMPLOYEE	....

PROJECT	EMPLOYEE

Assignment in the all key relation.

Relating this back to the DLS problem we note that the technique has produced one too many of these all key tables. To further illustrate the problem that DLS has in representing relationships of differing degrees, we now consider a ternary degree example.

1.2.8.3 Ternary relationships

1.2.8.3.1 Introduction

The traditional EMPLOYEE-PROJECT-SKILL_USED ternary degree problem [Teory et al. 1976] is examined.

1.2.8.3.2 DLS performance evaluation w.r.t ternary.

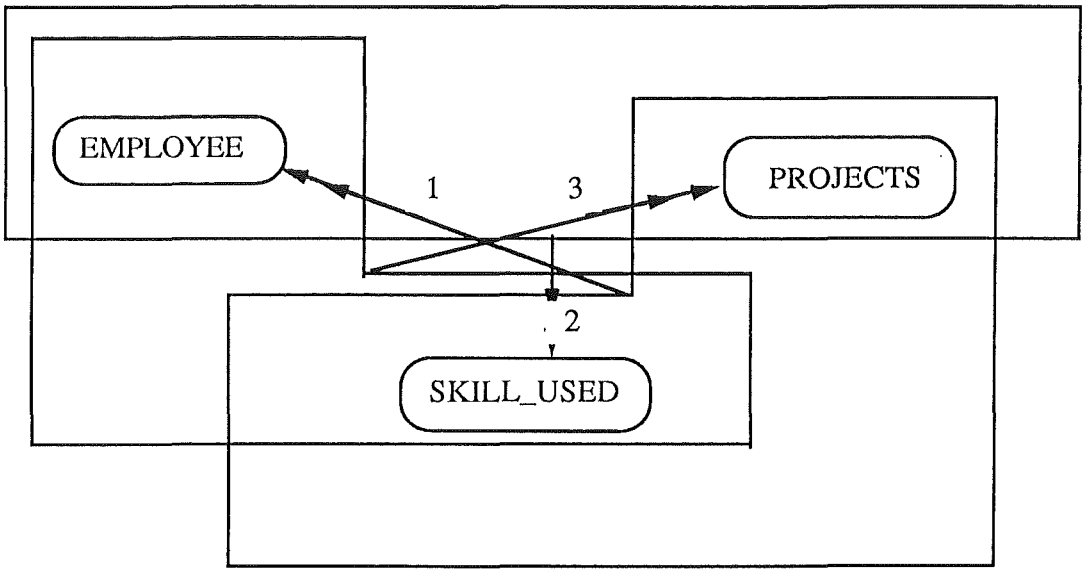
An employee will have many skills that he/she will use on different projects. However, it is not certain which skill an employee will use on any particular project.

Dependency list

1 - An EMPLOYEE uses many SKILL_USED on many PROJECTS.
2 - A SKILL_USED is used by many EMPLOYEEs on many PROJECTS.
3 - A PROJECT will have many EMPLOYEEs using different SKILL_USED.

Note that all of these are independent dependencies and that they can not be represented as one dependency statement, (as DLS stands at present).

Dependency diagram



(I apologise for this diagram, but doing this with bubbles is not the trivial task that it first appears to be)

Finished tables

SKILL_USED	EMPLOYEE	PROJECT

SKILL_USED	EMPLOYEE	PROJECT

SKILL_USED	EMPLOYEE	PROJECT

Decomposition dictates that the correct way to represent a ternary relationship, (or n-ary relationship), is to derive an all key table comprised of all of the participating data fields, (i.e. R(PROJECT, EMPLOYEE, SKILL_USED).

Again DLS has produced the correct table, but it has produced too many of them.

1.2.8.4 Identifying n-ary relationships.

It is appropriate at this stage to provide a guide on how to identify n-ary relationships, before any discussion on what to do about them.

There are two tell-tale signs of a n-ary relationship:

- Dependency lists that define dependencies among the same data fields. Generally each dependency is in the form of a data field being the subject of a combined dependency of all of the other data fields.
- A dependency diagram that is "cyclic", i.e. a diagram that has a data field which is part of a prime key, which is part of a prime key, ... , of itself. Note in the diagram that by tracking the prime keys of PROJECT, i.e. EMPLOYEE , (or SKILL_USED), and then examining the prime keys of EMPLOYEE, (or SKILL_USED), we find PROJECT.

In theory there will be n dependency list statements for an n-ary relation. In practise it is predicted that some to the dependency list statements may be overlooked in a high degree relationship.

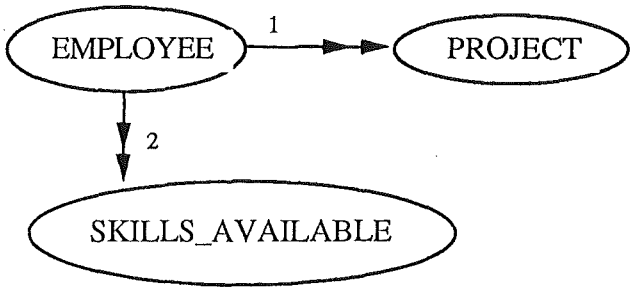
To emphasis this fact consider this scenario that may appear in the dependency list stage to be a ternary relationship.

Dependency list

1 - An EMPLOYEE works on many PROJECTS.
2 - An EMPLOYEE will have many SKILLS_AVAILABLE, (that can be used on all of the PROJECTS that he/she works on).

The bracketed part of statement 2 is extra information.

Dependency diagram



Hence we have correctly identified two separate MVDs in what appeared to be a ternary relationship. Notice that there is no cyclic structure.

1.2.8.5 Discussion of problems.

"N-ary relationship" is an inappropriate term to use in a dependency orientated method. Therefore let us define:

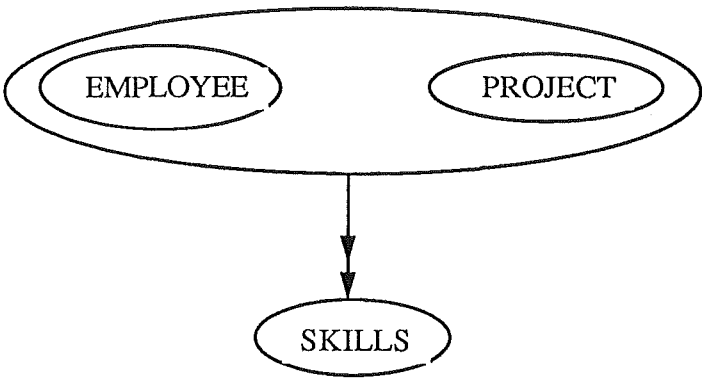
N-dependency: There exists an n-dependency among n data fields if every data field a combination of all the other n-1 data fields define values of that data field.

It is apparent that DLS does not correctly model an n-ary relationship. It may be argued that as only one of the all key tables are needed, the extra dependencies that are in the dependency list and diagrams could be omitted. In the given ternary example this would change the dependency list and diagram to

Dependency list

1 - An EMPLOYEE uses many SKILLS on many PROJECTS.

Dependency diagram

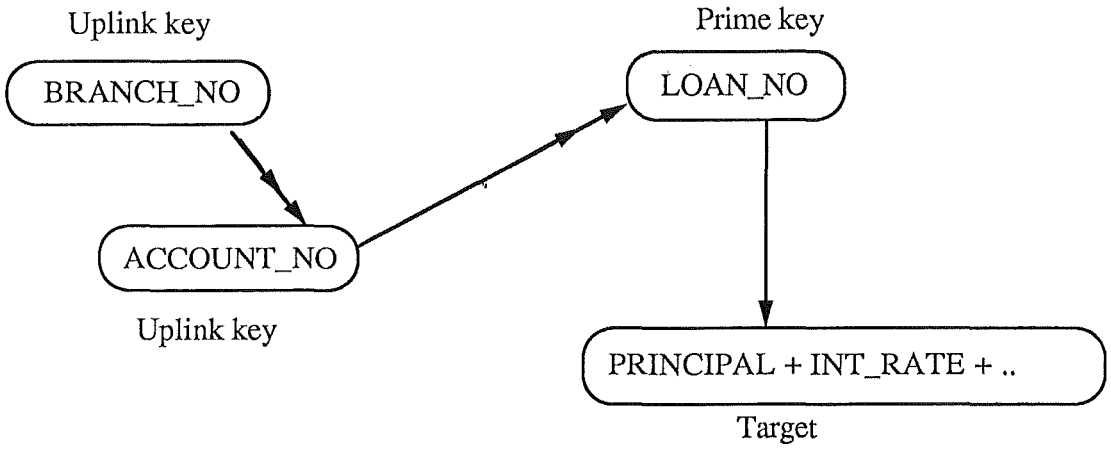


This will produce the correct finished table, but it is not a correct representation of the dependencies acting in the model! The dependency of SKILL on EMPLOYEE and PROJECT is only part of the whole story. Furthermore, this case could be easily mistaken for the case where there are two many:many relationships acting on the same attribute, (i.e. the SKILLS_AVAILABLE case as opposed to the SKILL_USED).

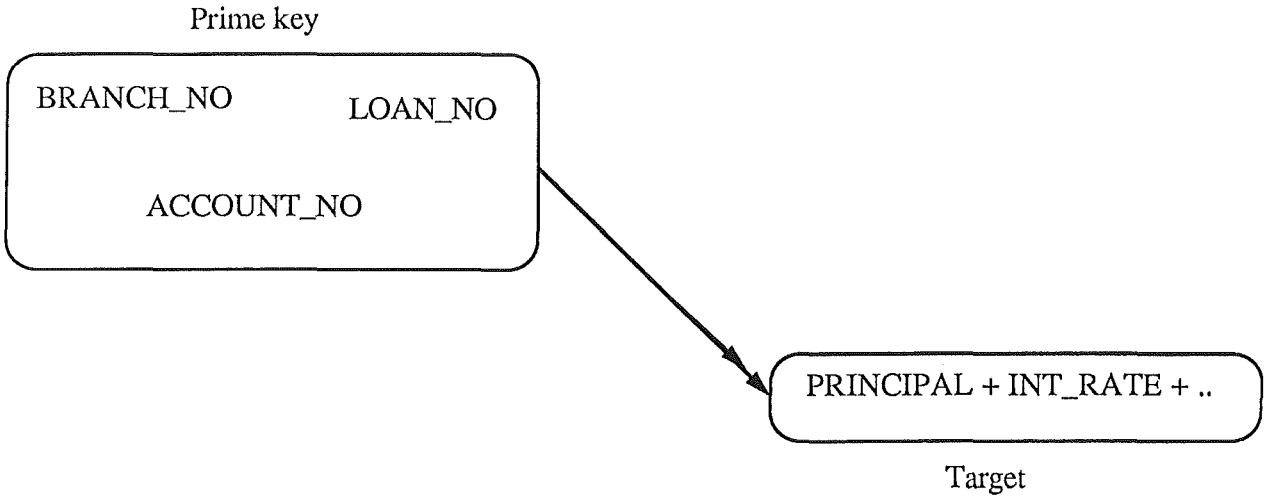
Therefore it is proposed that an alternative to the dependency diagramming process be implemented.

To introduce my proposal an extra piece of information must be stated.

It is possible to combine uplink keys and prime key fields into one prime key bubble in existing DLS.



To

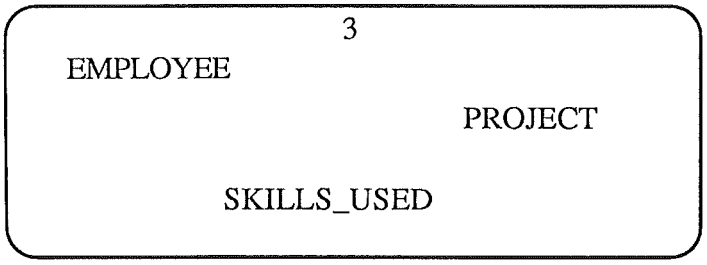


It is considered that this is an inappropriate course of action because the dependencies in between the data fields are lost. It has been shown that the order of dependencies IS important, and such a loss of information can cause loss of normal form.

1.2.8.6 Proposals.

It would therefore seem appropriate to reserve the Multivalued bubble for only n-ary dependencies. Thus the EMPLOYEE/SKILL_USED/PROJECT example of section 1.2.8.3.1, will have a correctly modeled corresponding Dependency diagram

Dependency diagram



Dependency lists will also have to change accordingly. The original list should be concatenated to one statement, by explicitly stating that there is a n-dependency between the participants. The dependency list statement number that defines the n-dependency should be included in the bubble for purposes of identification.

Finished tables

SKILL_USED	EMPLOYEE	PROJECT

For every n-dependency bubble a finished all key table of all the participating data fields should be produced.

1.2.8.7 Attributes of n-dependencies

Just as n-ary relations may have attributes, n-dependencies may have attributes also.

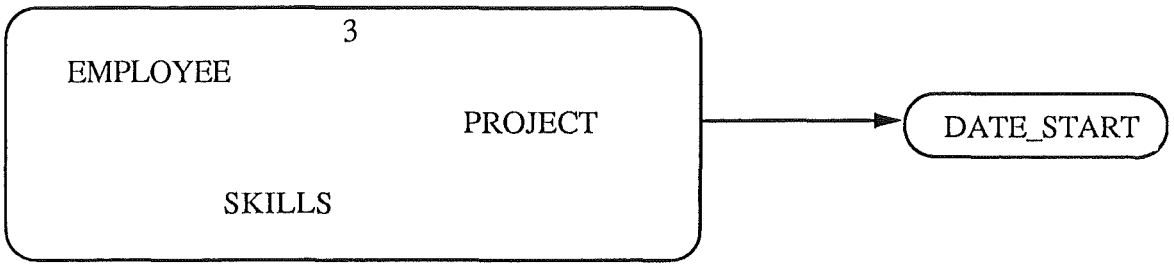
An n-dependency attribute is denoted as a SVD off of a n-dependency bubble.

An example of this is that if the starting date of when an employee first uses a skill on a project is necessary. This may be shown as

Dependency list

4 - The DATE_START of the n-dependency between EMPLOYEE and SKILL and PROJECT is stored.

Dependency diagram



It is recommended that a separate dependency list, (preferably straight after), other than the one defining an n-dependency be used.

Section 1.3.0 Summary.

This part of the project set about clarifying some issues that DLS raised. The hope was that interesting results would be found. I can conclude that results that will help DLS become a more accepted technique have discovered. The results can be split up into 2 parts:

1.3.1 Useful guidelines to the use of DLS

Potential problems that a designer might fall into, and guides that a designer may use are to be found in sections: 1.2.2.4, 1.2.4.3, 1.2.5.2, 1.2.6.5.4, 1.2.6.6.3, 1.2.7.2, 1.2.8.2, 1.2.8.5, and 1.2.8.6.

1.3.2 Academic discoveries made in DLS

There were discoveries as to how DLS reacts to certain conditions made in: all of section 1.2.3, 1.2.4.4, all of section 1.2.6 while examining the claim that DLS achieves 5NF, and n-ary relationships addition to DLS in 1.2.8.

PART 2: A real life application.

"Database (DB) design can be an extremely complex task (at least in a "larger" database environment; the design of "small" databases is usually straight forward " [Date 1986]

2.1 Introduction.

An academic survey by itself would hardly be a comprehensive evaluation of a design method. The purpose of design methods is to help produce usable databases for real world applications, and not only to provide academics with a source of argument. It was therefore decided to apply DLS to a real world application with the hope of providing some gauge of its level of effectiveness in producing a workable solution.

2.1.1 The problem.

The problem was introduced in section i.1 as the conversion of a system from one underlying data model to another while converting a system between two machines.. This process is termed "schema conversion" and is of interest in a practical application.

2.1.2 Schema conversion issues.

The conversion process, and the larger topic of schema integration, is a major practical problem [Batini et al. 1986]. The problem of converting from one database to another has been dealt with before - a prevalent case being the "upgrading" of CODASYL systems to a relational implementation. The practical issues can be viewed as trying:

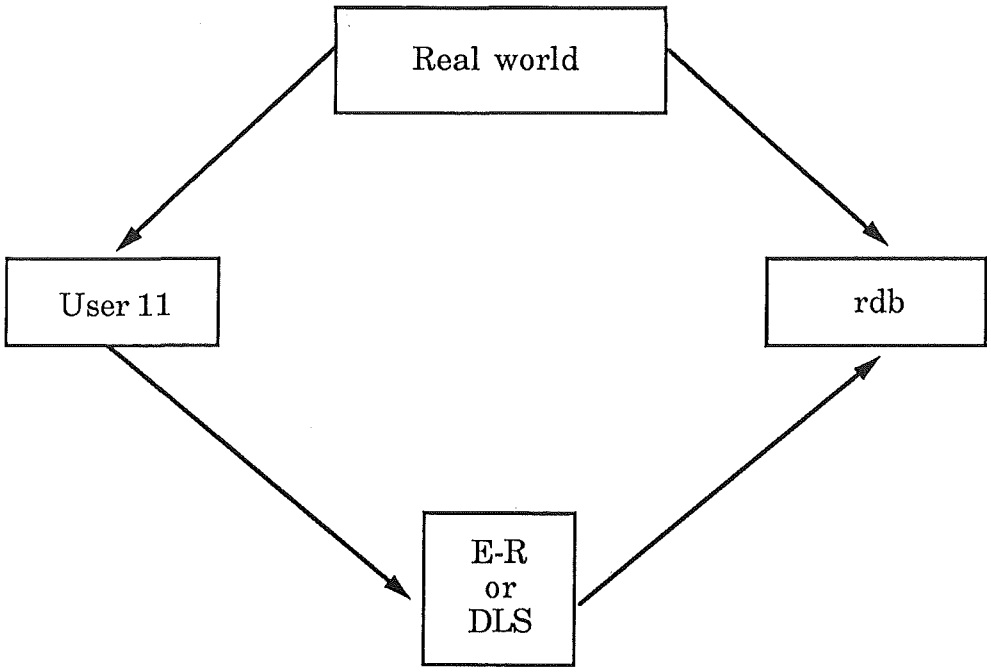
- to retain as much of the semantic and data dependency information of the enterprise as is possible, while
- constructing the necessary catalogue and bookkeeping information that is particular to the new model,
- while deleting all of the now redundant catalogue information that was used in the old model.

The scale of this problem may now be apparent.

The larger topic of schema integration is the merging of the fragmented schema - that often will be produced by a design technique - into one overall view of the data. The analysis may be done by functional areas, leading to several views which may overlap. The potential issues of schema conversion are that the techniques of producing a new converted schema - that is based on an already existing schema - are different from modeling an enterprise from scratch. There already exists a schema that can be analysed, and an impression of the necessary data requirements may possibly be derived from the analysis. How useful the existing schema is will not be known at the start of the process. Producing a completely new system from the beginning will involve a lot more person contact analysis, as probably no such enterprise overview will be available.

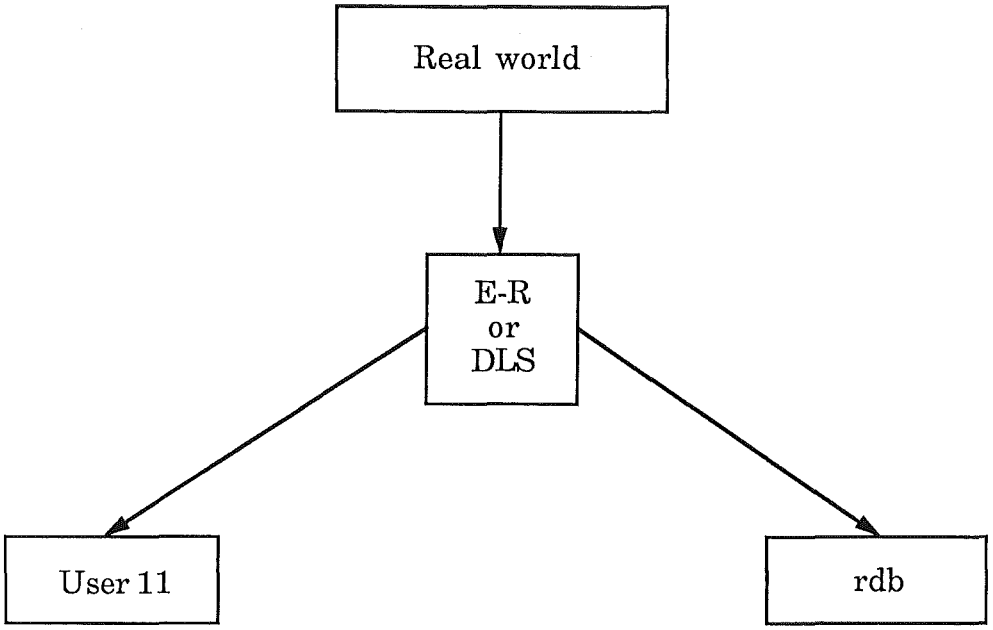
This may be represented diagrammatically as going through the extra three steps in the diagram of Real-world to User 11 to DLS, (or ER, etc.), to the rdb implementation, instead of going straight to the rdb database.

Modelling diagram



Not all of the original conceptual model may have been able to be implemented in User 11. The aim of the conversion is to deduce such information without repeating the original analysis.

Modelling diagram



Section 2.2 The analysis.

2.2.1 The system to be analysed.

The User 11 file handling system, (see section i.3.3.2), is not a true database model. It uses indexed ASCII character files that holds masked character data, hardcoded indexing, and data access routines within each Basic application program.

There are currently approximately 50 User 11 indexed file structures, which vary in size from 8 to 140 fields per record. Each of these files has relationships with at least one of the other files. There are 270 Basic programs associated with these files, giving a very complex assortment of data relationships to be analysed.

As User 11 is a file handling system, and not a database model, it does not have the "relational normalisation" that would be expected from a relational database. "Flat file" systems, such as this, can be in relational normalisation, but this one certainly is not. Due to the keying mechanisms that are necessary to run a file handling system, much duplication and redundancy of data has been discovered during the data analysis. This data redundancy and duplication will hopefully be removed when the new schema is designed.

2.2.2 The analysis performed.

The first step of DLS is to derive the dependencies between data fields. This first required a "coming to grips" period with the basic structure of User 11, and the CDB's data processing methods.

2.2.2.1 CDB's data processing methods.

The CDB use the Debit/Credit accounting structure. Each external company/business/private-client entity that the CDB transacts with, and each of the CDB's internal sections, has its own individual accounts.

A business transaction is represented by an invoice. Each invoice is "double accounted" where the relevant Debit and Credit accounts are offset so that the total balance of an invoice is zero. Invoices are passed through the system weekly, with transactions being archived for a period of three months. It is the User 11 system's job to process these invoices.

2.2.2.2 Details of the analysis.

There were two methods used to analyse the system:

- a bottom-up approach, and
- a top-down approach.

These methods strongly correspond to their counterparts in chapter 4 of the technical report. The bottom-up part required looking at the raw data elements, and the way that they all link together. The top-down part required studying the data flow through the system, and the high level structure that the system uses.

By combining the results of the 2 stages together, a greater understanding of the entire system, and its requirements, was achieved.

This analysis was performed to produce the dependency lists. Smith states in the original paper:

"The design of a proper database is of course impossible without a thorough understanding of the application's data requirements. The designer must know all data

fields required by the application and understand the single-valued and multivalued dependencies between those fields."

But no actual mention of a method on how to achieve this was given. It is therefore necessary to incorporate a data modelling technique, like bottom-up or top-down or both, to achieve this objective.

2.2.2.3 The Bottom-up analysis.

The User 11 system stores all of its information in ASCII files. The Basic application programs access records of these files to attain account information. All of the bookkeeping information is also stored in records of the same ASCII files. A very large variety of different field formats, (what will be the DLS's future finished tables representations), must be stored in a small, (only 50), number of ASCII record files. The system overcomes this by brutally "overlaying" the files. Therefore one file may have several different data formats stored in it, with each record having a flag representing which is the correct data format for that particular record. There was very little available documentation on the overlay structure.

The exact details of the application programs data access methods were virtually unfathomable. When an application program wanted to access a file it:

- first had to access the definition of the overlay mask for the correct field format, and
- then somehow calculate the correct record number, and
-finally access the correct record in the file.

It was not a trivial task to establish the dependencies between the data fields, especially with such a mysterious way of storing the dependency information of User 11.

2.2.2.4 Top-down analysis: the modular structure.

The system consists of 4 independent modules:

- General Ledger, and
- Creditors, and
- Debtors, and
- Stock.

Each module is a logical grouping of the ASCII files. A module is generally self contained, with several files that stores details for the intermodule communication. The creditors and debtor modules stores the comings and goings of the system, the stock module stores the stock that is held by the CDB, and the general ledger holds most of the bookkeeping information, (i.e. all the accounts, name information, etc).

2.2.3 The resulting dependency lists and diagrams.

The actual analysis that was performed is provides as appendices A, B, C, and D.

2.2.4 Areas of interest in the analysis.

During the CDB's analysis several interesting situations arose where I was able to use the recommendations of Part 1 to help model the situations correctly.

2.2.4.1 Attribute identification.

In the schema conversion process that was been undertaken, the step of separating the attribute type facts from relationship type facts was an invaluable aid. As the existing system has a completely inapplicable keying mechanism, (involving record overlays), the structuring of an elegant key structure was essential.

By examining the tree structure of the resulting Debtors dependency diagram, (Appendix C), several points can be made. With the relationship data fields positioned to the left, and the attribute data fields positioned to the right of the diagram, the data structure and dependency information is uniformly presented. Furthermore, by initially

identifying and concentrating on the relationship information the design process was eased significantly.

2.2.4.2 Correct dependency identification.

As evident in the Stock module dependency diagram, (Appendix D), the deriving of the correct dependencies in this module was not a trivial task. Defining the correct dependency information, so that the correct finished tables resulted, was a process that took great care.

The recommendations that are made in part 1, were useful in avoiding traps that a non-informed designer may have fallen into. Notice that a ternary relationship between STOCK_MOVEMENT_SEQUENCE_NO, W_HOUSE_CODE, and PART_KEY was identified and diagrammed as such, (for the dependencies pmm1 through pmm4).

A very useful guideline was the complete avoidance of diagrams with a "cyclic" nature, (see section 1.2.8.4). Whenever one arose, a mistake was guarantied to have occurred.

2.2.5 Results of the analysis

A fully integrated schema, that the CDB can use to produce a relational implementation of their current system, has been produced. The finished implementation is dependent on the future CDB's decision on which relational database, and application programs, are to be purchased. It is foreseen that further work with the CDB is possible, once this decision has been made.

Section 2.3 Results and conclusions.

2.3.1 CDB's benefits

This part of the project was useful for the CDB because:

- they have been given a potentially finished schema for their future database, and
- the analysis and resulting documentation produced has provided them with a useful summary of their processing needs for their future system.

The finished schema may alter depending upon what choice of application they make.

2.3.2 DLS's analysis benefits.

This part of the project was significant for the DLS study because:

- a solution, (of a kind), was available so a full systems analysis was not needed, (and would have taken far too long), and
- DLS has only been used as a design method, any application to conversion or integration is unknown. Thus the work is original - nobody knew if DLS is suitable for this, and
- the DLS technique has provided an efficient, (cost in man hours compared to quality of final results), solution for the CDB.
- the finished diagrams and documentation will be useful for the CDB, because it is in an understandable form that the programmer/analysis can use, and
- the connection between the theory and practice of the DLS technique has been made. A theoretical modes has been used to produce a workable solution for a "real world" problem.

These above result conclusions can go a long way in supporting the claim that my modified DLS technique is a useful tool in the "real world"

Part 3 Summary.

The project has provided useful results that are both: immediately useful for practitioners of DLS, and can be published as a tutorial guide/examination of DLS. A published version of this paper would be titled

"A tutorial guide to Smith's bubblediagram technique".

This will incorporate the results and recommendations of Part 1:

- the identifying of attribute entities,
- how DLS MVDs work,
- how DLS manages to represent semantic constructs,
- DLS's deficiencies with BCNF, and the guides to overcome them, and
- the proposal of introducing n-ary relationships.

The added weight of the "real life" application will be added to all of my proposals.

The style of this report hovers between narrative that usually applies to published investigative papers such as this, and the formal style that is suitable for a project report. Any published article resulting will of course be a lot shorter.

References.

[Armstrong 1974] W.W. Armstrong
'Dependence structures of data base relationships'
Info Processing 74, North-Holland Pub. Co, Amsterdam, 1974, pp 580 - 583.

[Batini etal. 1976] C. Batini and M. Lenzerini and S.B Navathe ;
'A comparative analysis of methodologies for Database schema integration.' ;
ACM Computing Surveys, Vol 18 (4), Dec 1986.

[Batini et al. 1986] C. Batini and M. Lenzerini and S. Navathe ;
'A comparative analysis of methodologies for Database schema integration'
ACM Computing Surveys, Vol 18 (4) , Dec 1986, pp 323 - 364.

[Bernstein 1976] P.A. Bernstein
'Synthesizing 3rd normal form relation from functional dependencies.'
ACM tran on DB system, 1, 277 - 298, Dec 1976.

[Bowers 1988] P. Bowers.
From Data to Databases
Van Wostrand Reinhold (Pub), 1988

[Chen 1976] Peter Pin-Shan Chen ;
'The Entity-Relationship ;model- toward a unified view of data'
ACM Transactions of Database systems, Vol 1 (1), March 1976
pp 9 - 36.

[Codd 1970] E.F Codd
'A relational model of data for large shared data banks'
CACM, 13 (6), June 1970, 377-387

[Codd 1971] E.F. Codd
'Further normalization of the Database relational model'
Courant Comp Sci Symposium 6, D/B System,
Prentise Hall Pub, New York, May 1971 , pp 65 - 98.

[Codd 1979] E.F. Codd
'Extending the database relational model to capture more meaning'
ACM TODS 4 (4), Dec 1979.

[Date 1986] Date C.J ;
An introduction to Database Systems ;
Vol 1 Fourth Edition.
Addison-Wesley Pub.

[Date 1983] Date C.J.
An Introduction to Database Systems
Vol 2 fourth Edition
Addison-Wesley (Pub), 1983

[Fagin 1977a] R. Fagin
'The decomposition versus the synthetic approach to relational database design'
Proc 3rd international conference on very large DB, 1977 , pp 441 - 446.

[Fagin 1977b] R. Fagin ;
'Multivalued Dependencies and a new normal form for relational Database'
ACM Transactions of Database systems, Vol 2 (3), Sept 1977
pp 262 - 278.

[Fagin 1981] R. Fagin
'A normal form for Relational Databases that is based on domains and keys'
ACM Trans. Database Systems, Vol 6 (3), Sept 1981, pp 387 - 415.

[Furtado et al. 1986] A. Furtado and E. Neuhold.
"Formal techniques for Database design"
Springer-Verlag (Pub), 1986

[Howe 1983] D.R. Howe
'Data analysis for DataBase design'
Edward Arnold (publishers) 1983

[Jajodia et al. 1984] S. Jajodia and P.A. Ng and R. Yeh
'Introduction to the special issue on the use of Entity-Relationship concepts in Databases and Related Software'
Journal of system software, Vol 4 (2), July 1984, pp 95 - 98.

[Kent 1979] W. Kent.
"Limitation of record based systems"
ACM TODS 4 (1), March 1979

[Kent 1981] W. Kent
'Consequence of Assuming a universal relation'
ACM Transactions of Database systems, Vol 6 (4), Dec 1981
pp 99 - 121.

[Kent 1983] W. Kent.
'A simple guide to five normal forms in relational Database theory'
Communications of the ACM, Vol 26 (2) , Feb 1983, pp 120 - 125

[Kent 1983a] W. Kent
'The Universal Relation revisited'
ACM TODS, Vol 8 (4) , Dec 1983.

[Kent 1984] W. Kent
'Fact-based data analysis and design';
Journal of system software, Vol 4 (2), July 1984, pp 99 - 121.

[Maier etal. 1984] D. Maier and J.D. Ulman and M.Y. Varchi
'On the foundations of the Universal Relation Model'
ACM TODS, Vol 9 (2), June 1984

[Martin et al. 1985] J. Martin and C. McClure
'Diagraming techniques for analysts and programmers'
Prentice-Hall, 1985.

[Navathe et al. 1986] S. Navathe and R. Elmasri and J. Larson
'Integrating user views in database design'
IEEE Computer Vol 19 (1).

[Roessel 1987] Jan W. Van Roessel ;
'Design of a spatial data structure using the relational normal forms'
International Journal of Geographical information system, Vol 1 (1), 1987, pp 33 - 50.

[Smith 1985] Henry C. Smith ;
' Database design: composing fully normalized tables from a rigorous dependency diagram'
Communications of the ACM, Vol 28 (8) , Nov 1985, pp 826 - 838.

[Teory et al. 1986] T. Teory and D. Yang and J. Fry
'A logical design methodology for relational databases using the Extended Entity-Relationship model'
ACM Computing Surveys, Vol 18 (2) , June 1986.

[Ullman 1982] J.D. Ullman
' The U.R. strikes back',
Proc 1st ACM SIGACT - SIGMOD Symposium on principles of Database systems, March 1982.

[Ullman 1983] J.D. Ullman
'On Kent's ' 'Consequence of Assuming a universal relation'',
ACM TODS Vol 8 (4) , Dec 1983.

[Unger 1987] E. Unger and C. Angaye
' A synthesis algorithm for a class of 4NF'
ACM SIGSMALL/PC NOTES, Vol 13 (4), Nov 1987.

Appendix A The General Ledger module.

The first 2 letters are a code as to which of the original files the lists were derived. There is a descriptive passage at the start of each new section of dependencies that help give an overview of the tasks that each section should perform.

Dependency list

This section holds all of the account information and stores a lot of information that can be described as miscellaneous. If there was a peace of independent information, it was bound to appear here. This has been abbreviated in the interest of brevity.

ps1 - The main name of the company is stored as the LOCATION_CODE (eg. "00"), and LOCATION_DESC, (eg. "Christchurch Drainage Board").

ps2 - The name record of each department is stored as DEPARTMENT_NUM and DEPARTMENT_DESC, (eg. "00", "creditors").

ps3 - There is information stored in flag records to Creditors.
Invoices are flagged as "approved" or "non-approved".

CODE = " " means approved, "0" means unapproved.

FLAG_DESCRIPTION = "approved", or "unapproved".

ps4 - There is stored the information for the General ledger cost centre descriptions. Held is COST_CENTER_ANALYSIS_CODE, COST_CENTER_DESCRIPTION.

ps5 - Salesmen record. A way of grouping a bunch of debtors, i.e. label them by the business that sells things. Stored is SALESMEN_CODE, SALESMAN_GROUP, SALESMEN_NAME, SALESMEN_ADDRESS_LINE_1 through SALESMEN_ADDRESS_LINE_4.

ps6 - General ledger "shortcut lookup code". This is a way for the user to only key in a 3 character code instead of the full account number. This comprises of the G/L_LOOKUP_CODE, G/L_ACCOUNT_NUM, G/L_ACCOUNT_DECS a short description of the account.

This is the section that stores all of the information that is used in creating and deleting accounts.

pg1 - The internal sections of the CDB are split up into sections. Each section is split up into "entities". Each entity has a ENTITY_NUMB, (1 through 6), and a unique ENTITY_CODE.

pg2 - Each entity has a ENTITY_DESC, a brief description of what the entity is.

pg3 - there is certain bookkeeping information. The information is:

NUMB_PER : The number of periods in a year.

CURR_PER_NEMB : The current, (accounting), period.

YEAR_START_DATE : The accounting start date of this year, (dd/mm/yy).

YEAR_END_DATE : The accounting finish date of this year, (dd/mm/yy).

AUTO/MANUAL_VOUCHER_NUM : Each new voucher that is entered into the system is assigned a unique number. The current number that will be assigned to the next voucher that enters the system is stored here.

POST_PAST_PERIODS : Can transaction be bracketed into groups for batch processing, (Y/N).

DIRECT_POST_FUTURE : Can forward date transaction, (Y/N).

pg4 - The total number of entities that there are in the system is the NUMB_ENTITYIES_ACCOUNT_CODE, ENTITY_NUMB must be in between 1 and this number.

pg5 - There will be NUMB_ENTITES_ACCOUNT_CODE, (which stores the number of internal section that the CDB currently has set up), instances of each of these pieces of data:

MSTR_ENTITY_NUMB : The number of the entity that we are storing information about.

ENITITY_FIELD_NAME : 8 character entity name.

MSTR_ENTITY_DESC : Description of the entity.

This is the main accounts section. All bookkeeping information that an account will have is stored in this section.

pa1 - Each account has a unique ACCOUNT_NUMBER.

pa2 - Each account has certain information that is stored about it:

ACCOUNT_DESCRIPTION : Brief description of the account, (25 chars).

ACCOUNT_TYPE : This will hold one of three values: L (liability), A (asset), B (balance sheet).

NOMAL_BALANCE : Whether in it a Credit or Debit account, (C/D).

LEVEL_1

.

. : Breakdown of account number by its entity sections.

.

LEVEL_7.

EXIST_LAST_YEAR : This account existed last year.

pa3 - Each account will have this accounting information:

LAST_YEAR_CLOSING_BALANCE : If this account existed last year.

YEAR_TO_DATE_BALANCE : This account current balance.

OPENING_BALANCE : The opening balance that this account had at the start of the year.

PERIOD_1_BAL

.

. : Break down of accounts by entity number.

.

PERIOD_13_BAL.

pa4 - If an account existed last year then there will also be this accounting information stored:

L_Y_LAST_YEAR_CLOSING_BALANCE : If this account existed last year.

L_YOPENING_BALANCE : The opening balance that this account had at the start of the year.

L_Y_PERIOD_1_BAL

. : Break down of accounts by entity number.

.

L_Y_PERIOD_13_BAL.

This section track the flow of a voucher. A voucher is a manual document showing the cheques going in and out of the CDB.

pp1 - A voucher will have a unique VOUCHER_NUM, (when a voucher request one of these the value is derived from AUTO/MANUAL_VOUCHER_NUM).

pp2 - A voucher may have many LINE_NUMBERS in it. Each line number is for a different line in the voucher.

pp3 - Each LINE_NUMBER will have certain details to store:

REF_SUBSIDARY_LEDGER : If this line number pertains to a creditor or supplier, (stored in POSTING_TYPE_CODE), the suppliers or creditors code number is stored.

POSTING_TYPE_CODE : a line in a voucher can be one of several different sorts, IN (invoice), DA (debit), CR (credit), CS (cash).

DECS_JOURNAL : A brief description of the transaction.

DATE_JOURNAL : The date that the transaction took place.

pp4 - Each LINE_NUMBER has the ACCOUNT_CODE that this line in the voucher is responsible/receiving payment..

pp5 - Each LINE_NUMBER has the POST_SOURCE_CODE stored which is the group that the account originates from.

This section stores bookkeeping information for each Voucher.

pv1 - for each VOUCHER_NUM there will be a payments made, for each payment against a voucher there is this information stored:

SEQ_NUM_#_IN_POSTING : The number of payments that this voucher has had against it.

FULL_G/L_ACCOUNT : The full General Ledger account number which is receiving/paying.

POST_VALUE : The value of this payment.

This section stores the information for each cheque that is posted to the General ledger system.

pq1 - Each Cheque that the system issues will have a CHEQUE_NUMBER. This is a unique number that the system creates, it is not the actual cheque number.

pq2 - For each cheque there is the PAYEE_NAME of the person/organisation that the cheque was issued to.

pq3 - For each cheque there is also this accounting information:

PAYMENT_DESC: A brief, (25 char), description of what the payment was for.

DATE_OF_CHEQUE : The date that the cheque was posted.

AMOUNT_OF_CHEQUE : The value that this cheque was made out for.

pq4 - For each cheque there is a PERIOD_NUM_OF_POSTING which is the accounting period that the cheque was issued.

This is the interface between the Debtors/Creditors/stock modules to the General Ledger. It contains data on all transaction, and cash for loading into the G/L. There should be 2 entries for each transaction: one for the relevant G/L job account, and one for the G/L control account.

ppst1 - A unique POST_REF_# is created to establish an index for each corresponding change in the stored information in the Stock/Creditors/Debtors that is entered through the system. A change in information will result when a

stock_movement/payment/debit is made. This therefore is part of the interface between the Stock/Creditors/Debtors modules and the General Ledger.

ppst2 - For each POST_REF_# there is a DEPT/BRANCH of the origin DEBT that this transaction involves.

ppst3 - For each POST_REF_# there is the G/L account code which is responsible/receives payment. This can either be a G/L_ACCOUNT_CODE or the shorter, (3 char), G/L_SHORT_CUT_CODE.

pps4 - For each POST_REF_# there is a POSTING_DATE, and POSTING_VALUE.

pps5 - For each POST_REF_# the module that it was created in is stored as SOURCE.

pps6 - For each POST_REF_# the POSTING_TYPE is stored which is the type of transaction that occurred, (i.e. AC (account credit), AD (account debit), SI (stock inwards).

pps7 - For each POST_REF_# there is a REFERENCE_NUM. This is the part number, or the invoice number, or the credit note number or the cheque number that is relative to the transaction.

pps8 - For each POST_REF_# the PERIOD_NUM that the transaction occurred in is stored.

pps9 - For each POST_REF_# there is a SOURCE_REF, which is the Customer/supplier for this transaction. These are SUPPLIER_CODE or CUST_CODE values.

Appendix B The Creditors Module.

The first 2 letters are a code as to which of the original files the lists were derived.

Dependency list

p1 - Each supplier that the CDB deals with is assigned a unique SUPP_CODE.

p2 - Each SUPP_CODE has this accounting bookkeeping information stored:
BALANCE : The suppliers current balance.
TOT_DEBT_PER : The total debits this period.
CURR_PER_BAL : The current periods balance.
PER_1_BAL .. PER_5_BAL : The balance for the relevant period.
PER_6_PLUS_BAL : The balance form 6 and over periods ago.
PUR_PER_TO_DATE : The total amount purchased, (and processed), off of invoices this period to date.
PUR_YEAR_TO_DATE : The total amount purchased, (and processed), for this year.

p3 - Each SUPP_CODE has:
SUPP_NAME : the business name of this supplier.
ADD_LINE_1 .. ADD_LINE_4 : The address for this supplier.
SUPP_PHONE : The suppliers phone number.
SUPP_FAX : The supplier fax number.
SUPP_REMARK : A brief description of the supplier.

p4 - For each SUPP_CODE there is a CREDIT_TRAN_SEQ and DEBIT_TRAN_SEQ which are the number of current credit and debit transactions for the supplier.

p5 - For each SUPP_CODE there is a DAY_FOR_STATEMENT, which is the day of the month that a statement will be sent out to the supplier.

ppm1 - Each cheque that is posted in the system has a CHEQUE_NUM

ppm2 - Each CHEQUE_NUM has certain information:
SUPP_CODE : The supplier number who this cheque is destined to.

CASH_PAY_DATE : Date in which the cheque was written, (if not automatically generated).

SYSTEM_ENTERING_DATE : The date that the cheque was processed through the system.

CASH_PAYMENT_TYPE : Cash or cheque or manual payment.

PAYMENT_REFERENCE : A description of the reason for payment.

PAYMENT_AMOUNT : Amount for cheque.

ALLOCATION_RUN_NUMBER : Most cheques are automatically created by a batch run of a process. This is the run number of the batch.

MANUAL_CHEQUE_INDICATOR : Whether the cheque was manually or automatically generated.

pp3 - For each cheque that is created the PERIOD_NUM in which the cheque was written takes place.

pp4 - Each cheque will be for a PURCHASE_DEPT.

pp5 - Each cheque will be from a G/L bank account, BANK_ACCOUNT_CODE.

pr1 - For each INVOICE_NUM and SEQUENCE_NUM there is certain information stored:

ALLO_DATE : The date that the payment was allocated.

ALLO_CASH : The amount that the payment was for.

TRANS_TYPE : The method of payment.

CHEQUE_NUM_REF : The cheque or credit note number.

pr2 - For each INVOICE_NUM and SEQUENCE_NUM if a part was supplied then there will be a SUPP_CODE of the supplier who supplied the part.

pr3 - For each INVOICE_NUM and SEQUENCE_NUM if the payment was by cheque then a ALLO_RUN_NUM will be stored.

pt1 - For each INVOICE_NUM there is a:

SUPP_NUM : This is the supplier that this invoice is related to.

TRANS_SEQ : This is the number of transactions that this supplier has done in this current accounting period.

pt2 - Each invoice that is received by the CDB is assigned a unique VOUCHER_NUM.

pt3 - Each INVOICE_NUM has certain attributes:

TRANSACTION_DATE : The date that the transaction took place.

DUE_DATE : The date that payment is due, (normally the 20th).

DESCRIPTION : A brief description of the invoice.

SUPP_REF : the suppliers own reference number. Not a key to any other field in this system.

GROSS_VAL : the total value of all of the voucher.

LAST_OPENING_BALANCE : This is the last opening balance for the supplier themselves.

HOLD : When the invoice data is entered it is initially put on hold until everything is validated, (manualy), for payment.

pt3 - Each VOUCHER_NUM will have a certain number of payments made against it. This is stored as ALLO_SEQ_NUM.

pt4 - Each VOUCHER_NUM has the INVOICE_DEPT stored. This is a system code that indicates whether this is an internal or external transaction, (this is seldom used).

pt5 - Each VOUCHER_NUM has the first payment made against it stored as G/L_POSTING_RECORD. These are valued of POST_REF_NUM.

Appendix C The Debtors module.

The first 2 letters are a code as to which of the original files the lists were derived.

Dependency list

This first section stores all of the information about the debtors accounts. All permanent bookkeeping information is kept here.

pc1 - Each customer that the CDB deals with is assigned a unique CUST_CODE. Each CUST_CODE has:

CUST_NAME : the business name of this supplier.
ADD_LINE_1 .. ADD_LINE_4 : The address for this supplier.
CUST_PHONE : The suppliers phone number.
CUST_FAX : The supplier fax number.
CUST_REMARK : A brief description of the supplier.

pc2 - Each CUST_CODE has a SALESMEN_CODE, which is used in the G/L to group transactions together. Also a LOCATION_CODE is stored which is a CDB zoning number.

pc3 - Each CUST_CODE there is this accounting information stored:

BALANCE : The customers current balance.
TOT_DEBT_PER : The total debits this period.
CURR_PER_BAL : The current periods balance.
PER_1_BAL .. PER_5_BAL : The balance for the relevant period.
PER_6_PLUS_BAL : The balance form 6 and over periods ago.
SALES_PER_TO_DATE : The total amount sold, (and processed), off of invoices this period to date.
SALES_YEAR_TO_DATE : The total amount sold, (and processed), for this year.

pc4 - Each CUST_CODE has a DEPT_NO for the CDB department.

pc5 - Each CUST_CODE has a DEBT_TRANS_SEQ_NUM. This is the number of transactions that are still current for this customer.

This section has the debit transaction masters. It holds data for all transactions, example invoice creditors etc. which each debtor has, including value, tax, date, discounts.

pr1 - For each CUST_CODE and TRANS_SEQ_NUM, (which defines a transaction which will appear on an voucher), there are certain dates stored:

TRANS_DATE : The date that the transaction took place.

SYSTEM_DATE : The date that the transaction was entered into the system.

DUE_DATE : The date that payment is due.

pr 2 - For each CUST_CODE and TRANS_SEQ_NUM, the GROSS_VALUE of the transaction is kept.

pr3 - For each CUST_CODE and TRANS_SEQ_NUM, The AMOUNT_ALLOCATED_TO_DATE is stored. This is the amount of payment that has been received.

pr4 - For each CUST_CODE and TRANS_SEQ_NUM, the original VOUCHER_NUM is stored.

pr5 - For each CUST_CODE and TRANS_SEQ_NUM, the G/L_POSTINGS_FIRST_REC_NUM is the number of the key that stores this information in the General Ledger.

pr6 - For each CUST_CODE and TRANS_SEQ_NUM, the LAST_PERIOD_BALANCE_OUTSTANDING is stored.

pr7 - For each CUST_CODE and TRANS_SEQ_NUM, defines a ALLO_SEQ_NUM. This is the number of allocation are made against this transaction, (in payment).

Debtors cash allocation file. This keeps a record each time a cash or credit is allocated against an invoice or Debit transaction.

psa1 - For each VOUCHER_NUM and ALLO_SEQ_NUM, which is the identification for payment against the transaction, there are the ALLOCATION_DATE and ALLOCATION_TYPE. The type is whether payment was by cash, cheque, or credit note.

psa2 - For each CUST_CODE and TRANS_SEQ_NUM, the SALES_DEPT of the payment is stored.

psa3 - Each CUST_CODE and TRANS_SEQ_NUM, defines a BATCH_NUM and LINE_NUMBER, this is the number of the receipt that is received, and the number of the relevant line. Again this is system generated.

Debtors cash receipts masters. This section holds data for all cash received for each debtor, amount, date, amount allocated etc.

pcs1 - For each BATCH_NUM + LINE_NUM the STATEMENT_CUST_# is stored as a reference back to the customer that the original transaction was conducted with.

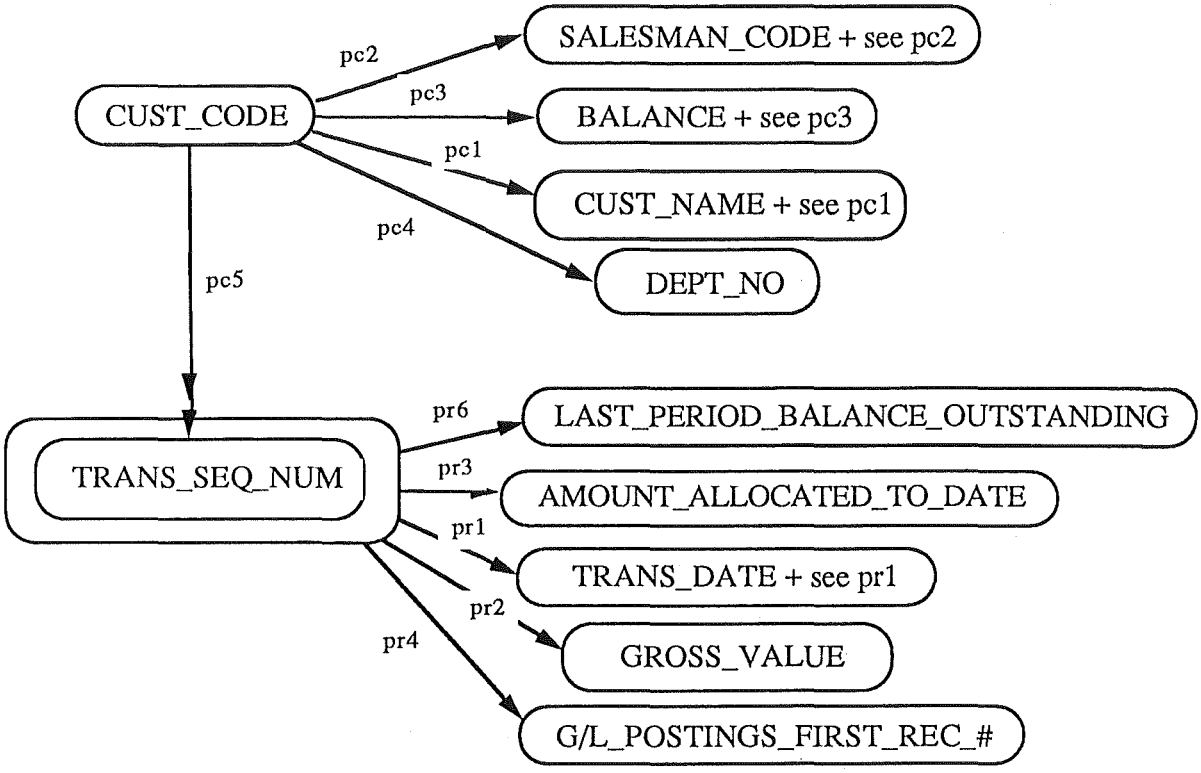
pcs2 - For each BATCH_NUM + LINE_NUM the CASH_RECIEPT_VALUE, and the SYSTEM_ENTRY_DATE is stored.

pcs3 - For each BATCH_NUM + LINE_NUM the CUST_DEPT is stored, this is either cash or cheque.

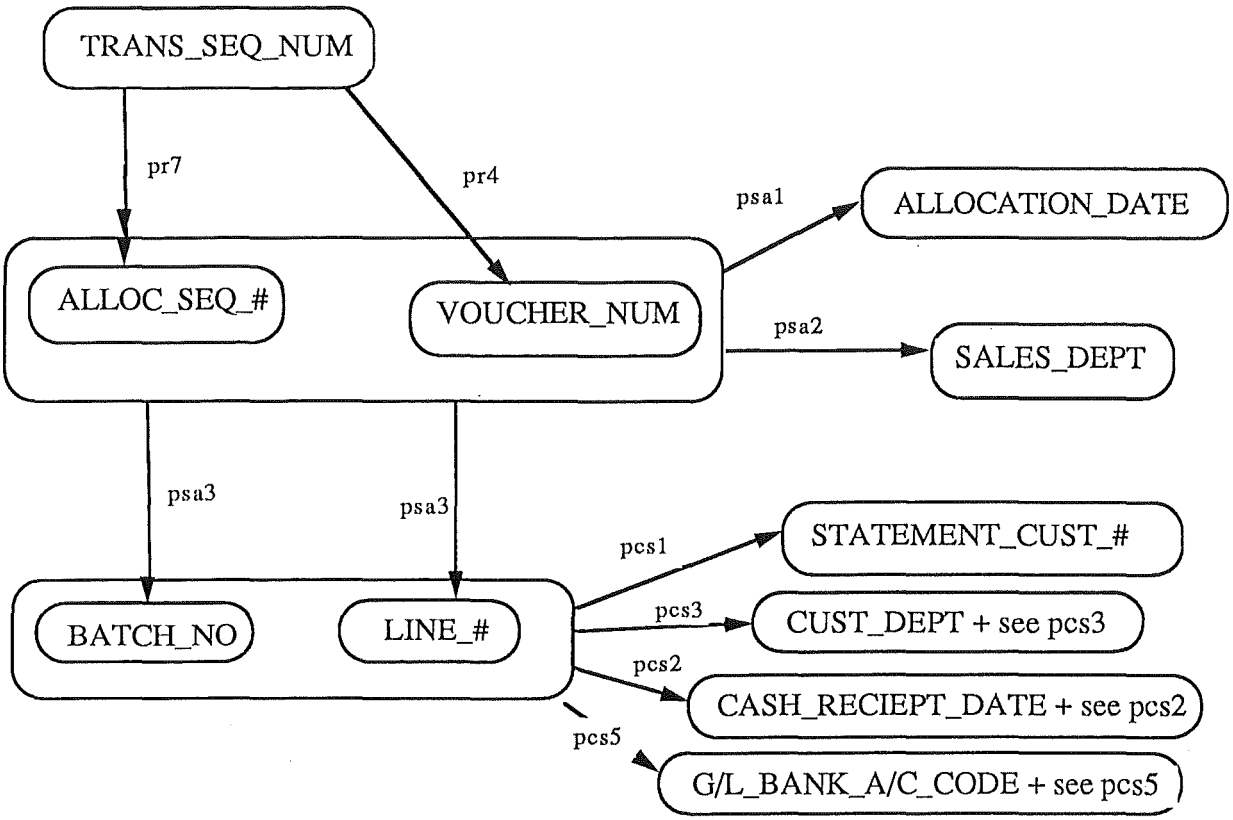
pcs4 - For each BATCH_NUM + LINE_NUM, the CASH_RECIEPT_VALUE and ALLO_TO_DATE, which is how much payment is received, is stored.

pcs5 - For each BATCH_NUM + LINE_NUM the G/L_SHORT_CUT_LOOKUP_CODE for the relevant department that payment will be received for, and G/L_BANK_ACCOUNT_CODE, which is the bank account that payment will be made to, is stored.

Dependency diagram



TRANS_SEQ_NUM is continued.


TRANS_SEQ_NUM
pr7
pr4
psa1
ALLOCATION_DATE
ALLOC_SEQ_#
VOUCHER_NUM
psa2
SALES_DEPT
psa3
psa3
pcs1
STATEMENT_CUST_#
BATCH_NO
LINE_#
pcs3
CUST_DEPT + see pcs3
pcs2
CASH_RECIEPT_DATE + see pcs2
pcs5
G/L_BANK_A/C_CODE + see pcs5

Appendix D The Stock module.

The first 2 letters are a code as to which of the original files the lists were derived.

Dependency list

This section contains all data referring to each part, desc, w/house, pricing, etc. Each part has its own record.

pp1 - Each PART_KEY, which is a unique identifier for each part, has a:

PART_DESC : A brief description of the part.

INVOICE_UNIT, SUPPLY_UNIT : When ordering or supplying these units, how many are ordered at once.

CONVERSION_FACTOR : conversion between invoice unit and supply unit.

MINIMUM_STOCK_LEVEL, MAXIMUM_STOCK_LEVEL : Used for automatically reordering the part.

SAFETY_STOCK_LEVEL : The bottom limit before reorder is necessary.

LOT_SIZE : The minimum number that are ordered at once.

QUANTITY_ON_HAND .

QUANTITY_ORDERED_THIS_PERIOD.

QUANTITY_ORDERED_YEAR_TO_DATE.

QUANTITY_INVOICED_TO_DATE.

VALUE_INVOICED_TO_DATE.

VALUE_ORDERED_YTD.

COST_OF_SALES_THIS_PERIOD.

COST_OF_SALES_TYD.

PURCHASE_LEAD_TIME : The amount of time that a order will take to arrive.

RETAIL_LIST_PRICE : The current price.

WEIGHTED_AVERAGE_COST : An average list price, taken over time. Recalculated every time one is ordered. Used to get an idea of what currently held stock is worth.

LAST_COST_PRICE : The cost of the last purchase.

pp3 - Each PART_KEY has a PRODUCT_GROUP_CODE.

pp4 - Each PART_KEY has the SUPP_CODE that supplies this part stored.

pp5 - Each PART_KEY there is information stored for the ABC calculations, i.e. the ABC_CODE.

pp6 - Each PART_KEY defines many STOCK_MOVEMENT_SEQ_# . This is a reference to a stock movement action.

pp7 - Information about the PART_KEY that is stored that is primarily used by the General Ledger are:

G/L_SALES_CODE : What account will be charged with the purchase of these goods.

G/L_STORE_ASSET_ACCOUNT : When a stock take is performed, what account will get the credit for any held stock.

As each part can stored in more than one w/house in more than one bin. The storage information is here.

ppb1 - A part can be stored in more that one warehouse, in more that one bin, this information is stored here. For each bin in a warehouse there is a PART_KEY the unique part identifier, W/HOUSE_CODE the code of the warehouse, and BIN_CODE the particular bin's code.

ppb2 Stored, and information are these particular details about a PART_KEY instance

QUANTITY_ON_HAND : The number stored in the bin.

STOCK_TAKE_BALANCE : the number in the bin at the last stock take.

ppb3 As there may be more bins with this part the NEXT_BIN and LAST_BIN is stored as a linking structure.

ppb4 The LAST_ACTIVITY_DATE is stored.

For each part in a warehouse the demand on the parts are recorded for ABC accounting.

ppx1 For each part in a warehouse, the demand on it is stored as: for each PART_NUM and W_HOUSE_CODE there is stored:

DEMAND_ABC_SUMMARY : The cumulative demand total.

DEMAND_PERIOD_1 : the demand for the last period.

Records movements of parts into and out of bins, example transfers, new stock, stock sold, etc. with dates quantities etc.

pmm1 Movement of stock are recorded around the system. Each KEY_CODE and W_H_CODE and unique REF_NUMBER, which can identify a stock movement, stores:

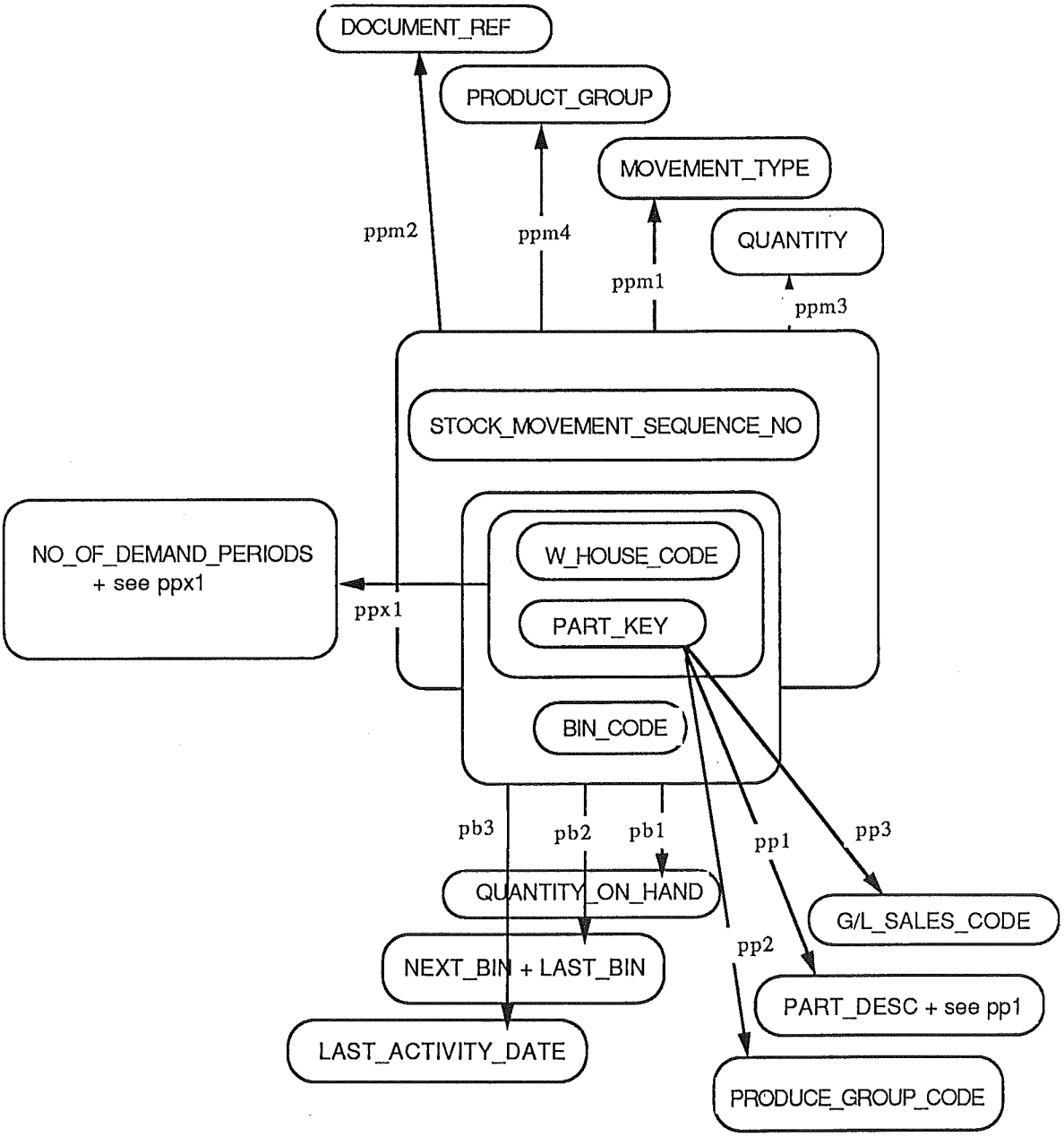
MOVEMENT_TYPE : There are several different types of stock movement.

ppm3 The requisition number, i.e. the piece of paper requesting the item is stored as DOCUMENT_REF

ppm4 The QUANTITY of items that were moved/supplied is stored.

ppm5 The BIN_CODE is stored for which BIN the items were taken.

Dependency diagram for Stock


DOCUMENT_REF
PRODUCT_GROUP
MOVEMENT_TYPE
QUANTITY
ppm2
ppm4
ppm1
ppm3
STOCK_MOVEMENT_SEQUENCE_NO
NO_OF_DEMAND_PERIODS
+ see ppx1
ppx1
W_HOUSE_CODE
PART_KEY
BIN_CODE
pb3
pb2
pb1
pp1
pp3
QUANTITY_ON_HAND
G/L_SALES_CODE
pp2
NEXT_BIN + LAST_BIN
PART_DESC + see pp1
LAST_ACTIVITY_DATE
PRODUCE_GROUP_CODE